OpenSSL

OpenSSL (https://www.openssl.org) és una llibreria lliure que ofereix una "API" criptogràfica de propòsit general als desenvolupadors d'aplicacions. És a dir, els permet incloure un conjunt de funcions criptogràfiques (ja provades i confiables) dins del codi font dels seus propis programes i d'aquesta manera no haver-se de preocupar dels detalls més tècnics sobre la seguretat de la informació, en delegar aquesta tasca en les funcions oferides per l'OpenSSL. També és una utilitat de línia de comandes que permet implementar directament aquesta API des del terminal, interactivament.

Tal com veurem, amb OpenSSL podem realitzar totes les tasques típiques de la criptografia (xifrat/desxifrat -tant simètric com asimètric-, creació/verificació de signatures, etc) així com de les relacionades amb la infraestructura PKI (creació/revocació de certificats, creació de CAs, etc) i altres funcionalitats extra, ja estiguin relacionades amb els protocols i algoritmes pròpiament TLS (estudi del "handshake") o més generalistes ("hashing", codificacions, etc).

> NOTA: Una eina alternativa a OpenSSL és GnuTLS (http://www.gnutls.org), la qual inclou igualment una llibreria i una utilitat de terminal (un petit turorial d'aquesta es pot consultar a https://raysnotebook.info/computing/crypto-tls.html) . Altres eines alternatives també són LibreSSL (http://www.libressl.org), BoringSSL (https://boringssl.googlesource.com) o wolfSSL (https://www.wolfssl.com; aquesta darrera especialitzada en funcionar sobre dispositius embeguts de poca capacitat de càlcul), entre d'altres. D'altra banda, també existeix la llibreria particular NSS (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS; la qual només és emprada per Firefox i obvia l'existència de les altres llibreries que puguin estar instal.lades al sistema). Totes elles són lliures i multiplataforma.

Un dels àmbits on l'ús de la llibreria OpenSSL és més visible és a l'hora d'implementar servidors segurs (normalment de tipus web) que xifrin i mantinguin la integritat de la comunicació amb els seus clients (i aconseguir així que aquest sigui segur pels seus usuaris, de manera que, per exemple, les seves visites no puguin ser "esnifades" per hackers, o que tinguin la garantia de no patir un atac "man-in-the-middle", etc). Això, a la pràctica, vol dir implementar servidors que facin servir el protocol TLS (el qual, recordem, "se situa per sobre" del protocol TCP -el nivell de transport al model TCP/IP- i "per sota" del nivell d'aplicació que correspongui) de forma que un servidor HTTP (per exemple) passi a ser HTTPS (HTTP + TLS). Per poder aconseguir això, una primera passa bàsica és crear un certificat (és a dir, una clau pública certificada per una autoritat) per després associar-lo a un servidor (normalment de tipus web però no té perquè) de la forma que aquest disposi a la seva configuració particular. La creació de certificats (els quals poden ser directament autosignats o bé signats per una CA externa a partir d'un CSR generat per nosaltres) pot ser realitzada perfectament per OpenSSL. De fet, amb OpenSSL fins i tot podríem implementar una CA pròpia per signar de forma centralitzada i autònoma els certificats dels diferents servidors que eventualment poguem gestionar.

> NOTA: Existeixen també diversos “wrappers” que faciliten la creació de certificats sense haver d'utilitzar les comandes que s'expliquen a continuació. Exemples són els programes que vénen amb OpenVPN (https://github.com/OpenVPN/easy-rsa) , servidors de correu, servidor Apache, etc. També són dignes de menció les següents eines similars de consola, encara que no es basin en OpenSSL: Cfssl (https://github.com/cloudflare/cfssl), Lemur (https://github.com/Netflix/lemur) o la comanda *keytool* que proporciona el paquet "openjdk-11-jre-headless"

Tasques habituals

La comanda *openssl* pot realitzar moltes tasques diferents, i és per això que incopora un seguit de subcomandes que serveixen per classificar aquestes tasques segons els seu àmbit. Per veure totes aquestes subcomandes es pot executar *openssl list -commands* Per veure els paràmetres que admet una subcomanda concreta, es pot executar *openssl help subcomanda* Però moltes vegades això no és suficient per saber totes les opcions que tenim disponibles perquè algunes subcomandes incorporen al seu torn vàries sub-subcomandes. Concretament *openssl list -digest-commands* mostra les sub-subcomandes relacionades amb la subcomanda *dgst* i *openssl list -cipher-commands* mostra les sub-subcomandes relacionades amb la subcomanda *enc* , entre d'altres.

> NOTA: Altres comandes openssl list interessants són *openssl list -cipher-algorithms*, *openssl list -digest-algorithms* o *openssl list -public-key-algorithms*

En qualsevol cas, tota la informació es pot consultar a les pàgines del manual d'Openssl, les quals, a més de la pàgina d'entrada *man openssl*, estan separades segons subcomandes, així: *man openssl-dgst*, *man openssl-enc*, etc

Respecte les subcomandes més importants que ens podem trobar, podem destacar:

openssl dgst ... : Tasques relacionades amb operacions de "hashing"
openssl passwd ... : Tasques relacionades amb operacions de "hashing" amb sals (per contrasenyes)
openssl enc ... : Tasques relacionades amb operacions de xifrat/desxifrat simètric
openssl rand ... : Tasques relacionades amb la generació de números aleatoris
openssl genpkey ... : Tasques relacionades amb la creació de claus privades
openssl pkey ... : Tasques relacionades amb la gestió de claus (canvi de formats, inspecció, etc)
openssl req ... : Tasques relacionades amb la creació i manteniment de CSRs
openssl x509 ... : Tasques relacionades amb la creació i manteniment de certificats x509
openssl verify ... : Tasques relacionades amb la verificació de certificats
openssl ocsp ... : Tasques relacionades amb la revocació de certificats (via OCSP).
openssl ca ... : Tasques relacionades amb la creació i manteniment d'una CA
openssl ciphers ... : Llista el conjunt de ciphersuites TLS existents, suportades, etc

A continuació s'indiquen alguns exemples pràctics de les subcomandes anteriors, a mode de "xuleta":

*Crear un "hash" (del contingut d'un fitxer o de *stdin*):

openssl dgst {-md5 | -sha512 | ... } [/ruta/fitxer]

NOTA: L'algoritme del "hash" s'indica després del paràmetre *dgst*. Pot ser qualsevol dels llistats a *openssl list -digest-commands* (precedit d'un guió)
NOTA: Si no s'especifica cap fitxer, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Es pot guardar la sortida en un fitxer amb el paràmetre *-out /ruta/hash.txt*
NOTA: Es pot signar el hash generat amb el paràmetre *-sign private.key* (suposant que tenim creada ja una clau privada, anomenada en aquest cas "private.key"); signar serveix per assegurar-se de què el hash no canvïi sense permís. Per tal de verificar-lo (suposant que el hash signat es va guardar, gràcies al paràmetre *-out*, en un fitxer anomenat "hashsignat.txt") cal usar els paràmetres *-verify public.key -signature /ruta/hashsignat.txt* (a més d'indicar al final la ruta del fitxer original)
NOTA: Existeixen "àlies" de les comandes "digest"; així, *openssl md5 ...* és equivalent a *openssl dgst -md5 ...* , *openssl sha512 ...* és equivalent a *openssl dgst -sha512 ...*

*Crear un "hash" amb una sal (d'una contrasenya per gravar-ho a l'arxiu "/etc/shadow", per exemple):

openssl passwd -6 [-salt unaSal] [contrasenya]

NOTA: El paràmetre *-6* indica que el hash serà de tipus SHA512 (el típic en Linux actualment). Altres números indiquen altres algoritmes
NOTA: Si no s'especifica cap sal, se'n generarà una aleatòria de 16 bytes.
NOTA: Si no s'especifica cap contrasenya, s'agafarà de *stdin* (i aquesta s'acabarà pulsant CTRL+D). També es podria indicar, si estigués escrita dins un fitxer (amb o sense salt de línia al final, això és irrellevant), amb el paràmetre *-in /ruta/fitxer*
NOTA: No existeix cap opció *-out* per guardar el resultat en un fitxer: sempre sortirà per la *stdout*

*Xifrar el contingut d'un fitxer amb un algoritme simètric:

openssl enc -e -XXX [-in /ruta/fitxer] [-out /ruta/fitxer.enc]

NOTA: Si no s'especifica cap fitxer d'entrada, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: El valor "-XXX" representa el nom de l'algoritme simètric escollit (precedit per un guió). Aquest nom pot ser un de la llista mostrada per *openssl list -cipher-commands* , com per exemple "**aes-256-cbc**" o (encara que no sigui tècnicament un algoritme de xifrat), "**base64**", entre molts d'altres.

NOTA: Aquesta comanda demana una "passphrase" interactivament que servirà per generar a partir d'ella internament la clau real utilitzada pel xifratge. Si el que es vol és indicar la "passphrase" directament a la pròpia comanda, cal indicar el paràmetre *-pass pass:hola* (suposant que la "passphrase" sigui "hola"). També es pot indicar així: *-pass env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-pass file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-pass stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*
NOTA: La sortida xifrada per defecte es genera en forma binària però es pot afegir el paràmetre *-a* per transformar el binari xifrat en texte ascii (que és més fàcil de gestionar). En realitat, el que fa *-a* simplement és codificar en Base64 el resultat obtingut del xifrat (i, en el cas del desxifrat, decodifica primer l'entrada)
NOTA: Per evitar el "warning" que es mostra en executar la comanda es pot afegir o bé el paràmetre *-pbkdf2* o bé el paràmetre *-iter nº*. Aquest darrer indica el número d'iteracions que patirà la "passphrase" introduïda per tal de derivar la clau d'encriptació realment utilitzada en l'algoritme de xifrat (a partir d'una sal en principi aleatòria però que també es podria escollir de forma fixa amb el paràmetre *-S salEnHex* , encara que això darrer en general en cal). Com major sigui el valor indicat en *-iter* més segura serà la clau però més costosa computacionalment serà.
NOTA: Es pot afegir el paràmetre *-p* per veure la sal i la clau generades per Openssl en aquest moment concret i que es faran servir per realitzar el xifratge actual
NOTA: Existeixen "àlies" de les comandes *openssl enc -e -XXX* ... amb la forma *openssl XXX -e* ...

*Desxifrar el contingut d'un fitxer amb un algoritme simètric:

openssl enc -d -XXX [-in /ruta/fitxer.enc] [-out /ruta/fitxer]

NOTA: Si no s'especifica cap fitxer d'entrada, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: El valor "-XXX" representa el nom de l'algoritme simètric escollit (precedit per un guió). Aquest nom pot ser un de la llista mostrada per *openssl list -cipher-commands* , com per exemple "**aes-256-cbc**" o (encara que no sigui tècnicament un algoritme de xifrat), "**base64**", entre molts d'altres.
NOTA: Aquesta comanda demana una "passphrase" interactivament que servirà per comprovar si la clau real utilitzada pel xifratge es correspon amb la generada en aquell moment a partir d'ella. Si el que es vol és indicar la "passphrase" directament a la pròpia comanda, cal indicar el paràmetre *-pass pass:hola* (suposant que la "passphrase" sigui "hola"). També es pot indicar així: *-pass env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-pass file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-pass stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*
NOTA: Si l'arxiu a desxifrar estigués en format ascii en comptes de en format binari, s'haurà afegir a la comanda anterior el paràmetre *-a* (per tal de decodificar primer aquest arxiu de Base64 a binari i llavors, un cop en format binari, procedir al desxifrat).
NOTA: Existeixen "àlies" de les comandes *openssl enc -d -XXX* ... amb la forma *openssl XXX -d* ...

*Xifrar el contingut d'un fitxer amb una clau pública (ja existent):

openssl pkeyutl -encrypt [-in /ruta/fitxer] [-out /ruta/fitxer.enc] -pubin -inkey public.key

NOTA: Si no s'especifica cap fitxer d'entrada, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: El paràmetre *-pubin* serveix per indicar que la clau especificada al paràmetre *-inkey* és pública (ja que per defecte *openssl* sempre la interpreta com a privada).
NOTA: Aquesta comanda no es fa servir gaire perquè no permet xifrar arxius més grans que el tamany de la clau pública emprada. De fet, on se sol fer servir és precisament per xifrar altres claus.

*Desxifrar el contingut d'un fitxer amb una clau privada (ja existent):

openssl pkeyutl -decrypt [-in /ruta/fitxer.enc] [-out /ruta/fitxer] -inkey private.key

NOTA: Si no s'especifica cap fitxer d'entrada, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: Aquesta comanda demanarà interactivament una "passphrase" (si la clau privada ha sigut creada amb ella) per tal de poder-la fer servir. Es pot indicar aquesta "passphrase" directament a la pròpia comanda amb el paràmetre *-passin pass:hola* (suposant que la "passphrase" sigui "hola"). També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té

com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*

*Signar el contingut d'un fitxer amb una clau privada (ja existent):

openssl pkeyutl -sign [-in /ruta/fitxer] [-out /ruta/fitxer.sig] -inkey private.key

NOTA: Veure totes les notes de la comanda anterior

*Verificar (i extreure) el contingut d'un fitxer amb una clau pública (ja existent):

openssl pkeyutl -verifyrecover [-in /ruta/fitxer.sig] [-out /ruta/fitxer] -pubin -inkey public.key

NOTA: Si no s'especifica cap fitxer d'entrada, l'entrada serà *stdin* (i aquesta s'acabarà pulsant CTRL+D).
NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: Veure https://github.com/openssl/openssl/issues/9658

*Confirmar que el fitxer signat es correspongui a un determinat fitxer original:

openssl pkeyutl -verify -in /ruta/fitxer -sigfile /ruta/fitxer.sig -inkey private.key

NOTA: Si la comprovació és exitosa, aquesta comanda mostrarà per pantalla el missatge "Signature Verified Succefully"; si no mostrarà el missatge "Signature Verification Failure"

*Generar un número pseudoaleatori:

openssl rand [-out /ruta/fitxer] nºbytes

NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: El paràmetre *-hex* fa que la sortida es mostri en format hexadecimal
NOTA: El paràmetre *-base64* fa que la sortida es mostri en codificació Base64.

---

*Crear una clau privada de tipus RSA (anomenada "private.key"):

openssl genpkey [-XXX] -algorithm rsa -pkeyopt rsa_keygen_bits:2048 [-out private.key]

NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: L'algoritme escollit s'indica amb el paràmetre *-algorithm*.
NOTA: El paràmetre *-pkeyopt* serveix per especificar les característiques tècniques de la clau a generar. En aquest exemple concretament s'indica el seu tamany, amb l'opció *rsa_keygen_bits*
NOTA: Una manera alternativa d'indicar certs tipus d'algoritmes de la clau generada és escriure una sèrie de paràmetres dins d'un fitxer (amb un format descrit a la secció "Parameter Generation Options" de *man openssl-genpkey*) i llavors utilitzar el paràmetre *-paramfile /ruta/fitxer.txt* de *openssl genpkey* (en comptes del paràmetre *-algorithm*). Per exemple, per crear una clau DSA el fitxer de paràmetres es podria escriure directament amb la comanda *openssl genpkey -genparam -algorithm dsa -pkeyopt dsa_paramgen_bits:2048 -out fitxer.param* i llavors es podria usar per crear la clau amb la comanda *openssl genpkey -paramfile fitxer.param -out private.key* No obstant, no farem servir aquesta manera de treballar perquè pel tipus de claus que farem servir no ens caldrà.
NOTA: Aquesta comanda no xifrarà la clau generada, a no ser que s'escrigui el valor "-XXX", el qual representa el nom de l'algoritme simètric escollit (precedit per un guió) per tal de xifrar la clau mitjançant una "passphrase" (la qual per defecte es demanarà a través de *stdin*, cosa que vol dir que es podrà indicar o bé interactivament en un terminal o bé a través d'una canonada). El nom concret de l'algoritme simètric pot ser un de la llista mostrada per *openssl list -cipher-commands* , com per exemple "**aes-256-cbc**" , entre molts d'altres.
NOTA: El fet de què la clau estigui xifrada implica que cada cop que la màquina hagi de fer-la servir (com per exemple, a l'hora de posar en marxa un servidor web segur) hagi de demanar la "passphrase" associada per desxifrar la clau i llavors poder-la utilitzar. Per tant, l'inconvenient de xifrar la clau amb "passphrase" és que cada cop que aquesta clau es necessiti fer servir, s'haurà d'introduir la "passphrase" d'alguna manera, i si aquesta manera és interactiva, fins que una persona no la escrigui el servidor en qüestió no s'iniciarà. Afortunadament, es pot indicar la "passphrase" (suposarem que és "hola") directament amb el paràmetre *-pass pass:hola* ; d'aquesta

manera ens estalviem haver-la d'introduïr interactivament. També es pot indicar així: *-pass env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-pass file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-pass stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl* . D'altra banda, l'inconvenient de no incloure cap "passphrase" a la clau privada és que si algú la robés de la nostra màquina i se l'emportaés a una altra, aquesta podria utilitzar-se com a impostora.
NOTA:Passphrases protect private keys from compromise when they are stored in backup systems but they don't help much in production because a knowledgeable attacker can always retrieve these keys from process memory; this is because, once activated, private keys are kept unprotected there. There are hardware devices (called Hardware Security Modules, or HSMs) that can protect private keys even in the case of server compromise, but they are expensive and thus justifiable only for organizations with strict security requirements. Thus, passphrases should be viewed only as a mechanism for protecting private keys when they are not installed on production systems. If you need better security in production, you should invest in a hardware solution
NOTA: Es pot afegir el paràmetre *-pkeyopt rsa_keygen_pubexp:65537* per indicar el valor de l'exponent usat per calcular la clau. Normalment això no cal perquè el valor per defecte ja és suficient.
NOTA: Es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format de la clau. Per defecte és PEM
NOTA: És important donar permisos restrictius a la clau privada generada de manera que només l"usuari "root" (o l'usuari amb el què estigui funcionant el servei que la necessiti utilitzar) puguin llegir-la (via *chown root:root* i *chmod 400* o *chown root:www-data* i *chmod 440*, o similars)
NOTA: Actualment, quasi tots els certificats públics són signats amb claus RSA de 2048 bytes. DSA és insegur i ECDSA no està suficientment suportat per les CAs encara (encara que amb el temps probablement veurem una pujada del seu ús).

*Crear una clau privada de tipus corba el·líptica P-256 (anomenada "private.key"):

openssl genpkey [-XXX] -algorithm ec -pkeyopt ec_paramgen_curve:prime256v1 [-out private.key]

NOTA: Si no s'especifica cap fitxer de sortida, la sortida serà *stdout*
NOTA: L'algoritme escollit s'indica amb el paràmetre *-algorithm*. En aquest cas, és "**E**l.liptic **C**urve"
NOTA: El paràmetre *-pkeyopt* serveix per especificar les característiques tècniques de la clau a generar. En aquest exemple concretament s'indica el seu nom l'opció *ec_paramgen_curve* Aquest valor determinarà el tamany de la clau així com altres característiques. Per saber els possibles noms a indicar com a valor d'aquesta opció es pot executar la comanda *openssl ecparam -list_curves* , la qual en mostra la llista.
NOTA: Llegeix les 3 notes de la comanda anterior relacionades amb als possibles algoritmes de xifrat que es poden indicar com a valor "-XXX" i de les implicacions que té afegir una "passphrase" (i les maneres de fer-ho).
NOTA: Es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format de la clau. Per defecte és PEM
NOTA: Una manera alternativa de crear una clau el·líptica equivalent es mitjançant la combinació de comandes següents: *openssl ecparam -genkey -name prime256v1 | openssl ec [-XXX] -out private.key* Només amb la comanda *openssl ecparam -genkey -name prime256v1 -out private.key* també la podríem generar però llavors no tindríem la possibilitat de xifrar (és a dir, d'indicar una "passphrase" a) la clau generada.

*Eliminar la passphrase d'una clau privada ja existent (útil per no haver d'escriure-la cada cop):

openssl pkey -in private.key -out newPrivate.key

NOTA: Aquesta comanda pregunta la "passphrase" a eliminar interactivament. Per escriure-la (suposarem que és "hola") directament en la comanda, s'ha d'indicar el paràmetre *-passin pass:hola* També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl.*
NOTA: Aquesta comanda també es pot fer servir pel contrari: per afegir una "passphrase" a una clau que no en tenia. Per fer això cal utilitzar els següents paràmetres (suposant que la "passphrase" és "hola"): *-XXX -passout pass:hola* on "XXX" representa el nom de l'algoritme simètric escollit (precedit per un guió), el qual pot ser un de la llista mostrada per openssl list -cipher-commands , com per exemple "aes-256-cbc" , entre molts d'altres.
NOTA: Es pot afegir el paràmetre *-inform {PEM|DER}* per especificar el format de la clau d'entrada (per defecte és PEM). D'altra banda, es pot afegir el paràmetre *-outform {PEM|DER}* si es vol especificar un format concret per la clau de sortida (per defecte és PEM).

*Comprovar les característiques d'una clau privada:

openssl pkey -in private.key -check -noout

NOTA: El paràmetre *-check* serveix simplement per mostrar el missatge "Key is valid" (o "Key is invalid")
NOTA: El paràmetre *-noout* serveix per no mostrar el contingut de la clau en sí
NOTA: Es pot afegir el paràmetre *-text* per obtenir més metainformació sobre la clau
NOTA: Aquesta comanda demanarà interactivament una "passphrase" (si la clau privada ha sigut creada amb ella) per tal de poder-la fer servir. Es pot indicar aquesta "passphrase" directament a la pròpia comanda amb el paràmetre *-passin pass:hola* (suposant que la "passphrase" sigui "hola"). També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*
NOTA: En el cas de claus el.líptiques la comanda anterior és equivalent a *openssl ec -in private.key -check -noout*

---

*Crear a la vegada una clau privada RSA i un CSR (anomenat "server.csr") signat per ella:

openssl req -new -newkey rsa:2048 -nodes -keyout private.key -out server.csr

NOTA: El paràmetre *-new* indica que es vol crear un nou CSR ("Certificate Signing Request")
NOTA: El paràmetre *-newkey rsa:2048* genera una clau privada de tipus RSA de 2048 bits Si només s'indiqués *-newkey rsa* (ometent el tamany), s'usaria el tamany per defecte indicat a l'arxiu de configuració d'OpenSSL.
NOTA: El paràmetre *-nodes* indica que la clau privada no estarà xifrada. Si no s'escrigués aquest paràmetre, la clau s'encriptaria amb una contrasenya ("passphrase") que es demanaria interactivament en aquell moment; això faria que cada cop que la màquina l'hagués de fer servir (com per exemple, a l'hora de posar en marxa un servidor web segur) demanaria aquesta contrasenya per desencriptar-la, cosa que no és molt pràctic. L'inconvenient de no encriptar la clau privada és que si algú la roba de la nostra màquina i se l'emporta a una altra, aquesta podrà utilitzar-se com a impostora.
NOTA: Si es vol xifrar la clau privada amb un algoritme robust, es recomana crear-la sense xifrar (fent servir el paràmetre *-nodes*) i llavors, posteriorment, afegir-li el "passphrase" amb la comanda *openssl pkey* ... adient (mostrada als paràgrafs anteriors). Això és perquè l'algoritme de xifrat que fa servir *openssl req* ... no és el més segur i aquesta comanda no té cap opció per canviar-lo.
NOTA: El paràmetre *-keyout* serveix per indicar el nom del fitxer on es guardarà la clau privada generada
NOTA: El paràmetre *-out* serveix per indicar el nom del fitxer (amb extensió .csr) que representa el CSR a enviar
NOTA: Es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CSR per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida)
NOTA: Es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CSR Per defecte és PEM

Recordem que un CSR és un document (contenint bàsicament una clau pública i signat amb la clau privada pròpia per garantir l'autenticitat) que representa una petició formal a una CA per a què aquesta el signi al seu torn amb la seva clau privada (i així donar fe de què aquesta clau pública es correspon a qui diu que és el propietari, el que a la pràctica es tradueix en obtenir-ne un CRT). La comanda anterior demanarà interactivament certa informació sobre l'individu/empresa/organització que vol fer aquesta la petició (és a dir, l'entitat origen del CSR). Aquesta informació, que es coneix com a "Distinguised Name (DN)", és guardada a l'interior del CSR generat per tal de, un cop hagi sigut enviat a la CA signant escollida, pugui ser fàcilment consultada per aquesta.

Un camp important del DN és el "Common Name (CN)", que **hauria de ser el nom DNS completament qualificat (FQDN) del servidor on s'utilitzarà el certificat** (si aquest servidor no tingués un FQDN, es pot indicar la seva adreça IP estàtica com a CN però no es recomana). Els altres elements del DN proporcionen informació administrativa addicional sobre el certificat (com l'empresa propietària del certificat, la ciutat i país on està registrada oficialment, etc) que és important per la CA però no és rellevant a nivell informàtic.

NOTA: Sempre que sigui opcional, es pot fer que alguna de les dades preguntades interactivament per la comanda anterior estigui buida indicant el valor punt ("."), ja que si es pulsa "Enter" s'assigna el valor per defecte existent a l'arxiu de configuració d'OpenSSL.

NOTA: La "challenge password" és un camp que també demana la comanda anterior però es recomana deixar en blanc. Va ser pensat per usar-se durant una revocació de certificats com una manera d'identificar l'entitat propietària del certificat (si s'indica, es grava tal qual dins del CSR per tal de posar-lo a disposició de la CA) però actualment és una dada que molt poques CAs implementen.

¡MOLT IMPORTANT!: el nom DNS (o direcció IP) que haguem introduït com a valor del camp "Common Name (CN)" NO PODRÀ CANVIAR DESPRÉS. Això és degut a què els certificats estan estretament vinculats al nom de la màquina, i si aquest canvia el certificat deixarà de ser vàlid.

> NOTA: Si volem que el certificat no només sigui vàlid per "www.elmeudomini.com" (per exemple) sinò també per qualsevol nom DNS del tipus "*qualsevolcosa*.elmeudomini.com", s'haurà d'escriure "*.elmeudomini.com" com a valor del "CN". Això és el que s'anomena certificat "wildcard". Compte perquè un certificat "wildcard" no cobreix (és a dir, no és vàlid per) el cas particular "elmeudomini.com"; si es volgués afegir, caldrà llavors utilitzar certificats SAN, que s'explicaran en el següent quadre gris. Un certificat "wildcard" tampoc cobreix noms amb més d'un nivell, del tipus "*unacosa.unaaltracosa*.elmeudomini.com"

> NOTA: ¿Com comprova el client si el certificat rebut del servidor conté efectviament el nom DNS del servidor (ja sigui específicament o a través d'un nom "wildcard") al que precisament vol accedir (i, per tant, té la garantia d'iniciar una connexió segura) si qualsevol connexió sempre es fa a posteriori de la consulta DNS (és a dir, que sempre s'accedeix a qualsevol servidor a través de la seva IP)? En el cas dels clients web (el més habituals amb diferència) aquest problema es solventa amb la capçalera de client HTTP "Host:": un cop el client ha realitzat la petició DNS i ha obtingut la IP del servidor (web) al què es vol connectar, afegeix el seu nom DNS a la capçalera "Host:" de la petició HTTP. D'aquesta manera, a més de saber el servidor quin "virtualhost" ha de mostrar (si en té varis disponibles), també sabrà quin certificat concret ha d'oferir (si en té varis). El client llavors comprovarà que el CN incrustat dins el certificat rebut coincideixi (ja sigui específicament o via "wildcard")amb el valor de la capçalera "Host:" enviada.

Com ja sabem, un cop tinguem el CSR, aquest s'haurà d'enviar a la CA escollida (via mail, formulari o un altre procediment que la CA en particular ofereixi) per a què ens retorni el fitxer ".crt" (és a dir, el nostre certificat signat per aquesta CA) llest per implementar-lo al nostre servidor.

Ens podem estalviar totes les preguntes interactives que fa la comanda *openssl req ...* afegint el paràmetre *-subj "/C=US/ST=New York/L=Brooklyn/O=Company/OU=Department/CN=www.example.com"* (especificant els valors concrets adients per cada cas).

> NOTA: El país s'ha d'indicar amb un "country code" de dues lletres; la llista completa es pot veure aquí: https://www.digicert.com/ssl-certificate-country-codes.htm

Hi ha una alternativa, però, per no haver d'omplir el "DN" interactivament sense haver de fer servir el paràmetre *-subj* anterior, i és editar directament l'arxiu de configuració d'OpenSSL, de tipus INI, per indicar-hi allà els valors per defecte desitjats dels camps "Country", "State", "Locality", "Organization", "Common Name", etc. Aquest arxiu s'anomena "**/etc/pki/tls/openssl.cnf**" a Fedora i "**/etc/ssl/openssl.cnf**" (a Ubuntu). De totes formes, a la pràctica, per no modificar l'arxiu general, el que es fa és crear un arxiu diferent, creat "ad-hoc", que contingui (només) les directives i els seus valors que volguem que siguin diferents dels valors per defecte indicats a l'arxiu "openssl.cnf" (especificant, això sí, les seccions on pertany cada directiva). I llavors indicar que a la comanda *openssl req ...* es farà servir aquest arxiu "ad-hoc" (que anomenarem per exemple "lameva.cnf") mitjançant el paràmetre *-config /ruta/lameva.cnf*

Concretament, els valors que ens interessaran indicar per automatitzar la creació d'un CSR són unes quantes que pertanyen a una secció el nom de la qual ha de venir indicat a la directiva "distinguished_name" (a l'arxiu general aquest nom és "[req_distinguished_name]") pertanyent al seu torn a la secció "[req]". Aquestes directives estan documentades extensament als apartats "Configuration File Format" i "Distinguished Name i and Attribute Section Format" de *man openssl-req* però un exemple de fitxer "lameva.conf" ad-hoc mínim podria ser, per exemple, el següent...:

```
[req]
distinguished_name = dnCSR
prompt = no
[dnCSR]
C = US
ST = New York
L = Brooklyn
O = Company
OU = Department
CN = www.example.com
```

...on la directiva "prompt" serveix, si val "no" (valor molt recomanable!), per poder indicar els valors desitjats en un format més automàtic del normal, valors que s'indicaran al seu torn a les directives "C","ST","L","O","OU" o "CN", de significat evident. Un cop gravat el fitxer "lameva.cnf", podrem generar el mateix CSR amb la comanda: *openssl req -new -newkey rsa:2048 -nodes -keyout private.key -out server.csr* ***-config lameva.cnf***

NOTA: Hi ha més directives que es poden indicar a la secció on s'especifiquen els valors del DN. Aquestes directives, opcionals serveixen per completar més informació sobre l'entitat que genera el CSR. La llista completa es troba al RFC5280 però alguns exemples són *emailAddress* , *name* o *surname* D'altra banda, dir que els valors del DN també es poden especificar en la seva forma "llarga", així: countryName, *stateOrProvinceName*, *localityName*, *organizationName*, *organizationUnitName* i *commonName*.

Una altra directiva interessant de conèixer a l'hora de generar CSRs és "req_extensions", pertanyent a la secció "[req]" del fitxer de configuració. El valor d'aquesta directiva és el nom de la secció sota la qual s'indicaran les diferents extensions que es vol que tingui el CRT final (a l'arxiu general aquesta directiva per defecte no s'utilitza) . D'entre les extensions que poden afegir a un CRT "pelat" (en parlarem de seguida) n'hi ha una, però, de molt important, l'extensió "**subjectAltName**".

Aquesta extensió fa que el certificat que es generi sigui vàlid no només pel nom DNS indicat a la directiva "Common Name" sinó que a més ho sigui per qualsevol altre nom DNS que s'especifiqui en una llista (dins, precisament, de l'extensió "subjectAltName)". Els certificats amb aquesta característica "multidomini" s'anomenen certificats "SAN" (de "SubjectAltName") i avui dia són els més habituals perquè permeten protegir amb un sol certificat multitud de noms DNS diferents (sense ells caldria tenir un certificat "clàssic" diferent per cada nom DNS diferent que volguem cobrir).

NOTA: Una diferència important entre els certificats SAN i els certificats "wildcard" és que els primers cobreixen (és a dir, "són vàlids per") diferents noms DNS que no tenen res a veure (per exemple, "www.pepe.com" i "ftp.manolo.org") mentre que els segons només cobreixen diferents noms DNS de primer nivell però que comparteixen un domini base comú (per exemple, "www.pepe.com" i "ftp.pepe.com"). Com a contrapartida, els certificats "wildcard" cobreixen d'entrada noms DNS que no han de predeterminar-se prèviament (el "*" serveix per tot) mentre que amb els certificats SAN s'ha de saber des del principi quins noms DNS s'afegiran a la llista per generar el certificat corresponent; això vol dir que si a posteriori volguéssim modificar aquesta llista (afegint-ne, eliminant-ne o modificant-ne noms DNS), caldria tornar a generar un nou certificat (amb el cost, logístic i potser econòmic, que això suposa).

Un exemple de fitxer "lameva.conf" ad-hoc mínim però incorporan l'extensió "subjectAltName" podria ser, per exemple, el següent...:

```
[req]
distinguished_name = dnCSR
req_extensions = extensionsCSR
prompt = no
[dnCSR]
C = US
ST = New York
L = Brooklyn
O = Company
OU = Department
CN = www.example.com
emailAddress = admin@example.com
[extensionsCSR]
subjectAltName = DNS:www.example.com,DNS:other.example.org,DNS:example.com,DNS:*.a.net
```

...on s'han indicat diversos noms DNS que el certificat cobrirà, com ara el del domini principal ("example.com"), un nom de primer nivell penjant d'aquest domini principal ("www.example.com"), un altre nom de primer nivell però penjant d'un altre domini principal diferent ("other.example.org") i, fins i tot, un nom wildcard penjant d'un altre domini principal diferent dels altres dos ("*.a.net").

NOTA: En un certificat SAN la directiva CN s'ignora completament (tal com marca el RFC2818). Així doncs, a la pràctica, en un certificat SAN s'han d'incloure tots els noms desitjats sota l'extensió "subjectAltName" i el valor del CN serà irrellevant

Existeix una sintaxi alternativa per indicar les extensions que és més fàcil de llegir ja que crea una secció específica per l' extensió en particular que es desitji. Així, l'exemple anterior es podria escriure d'aquesta altra manera:

```
[req]
distinguished_name = dnCSR
req_extensions = extensionsCSR
prompt = no
[dnCSR]
C = US
ST = New York
```

```
L = Brooklyn
O = Company
OU = Department
CN = www.example.com
emailAddress = admin@example.com
[extensionsCSR]
subjectAltName = @seccioParticular
[seccioParticular]
DNS.1=www.example.com
DNS.2=other.example.org
DNS.3=example.com
DNS.4=*.a.net
```

A més de la directiva *DNS.x* (on "x" representa un número sencer), en una secció de tipus *subjectAltName* poden haver més directives, com ara *IP.x* (per poder indicar IPs en comptes de noms DNS), *email.x* (per poder indicar direccions de correu) o *URI.x* (per poder indicar URIs), entre d'altres que es poden consultar a https://tools.ietf.org/html/rfc5280

En qualsevol cas, un cop gravat el fitxer "lameva.cnf" amb l'extensió "subjectAltName" afegida (sigui amb una sintaxi o amb una altra), ara podrem generar el CSR corresponent amb la mateixa comanda que vam fer servir per generar un CSR sense extensions: *openssl req -new -newkey rsa:2048 -nodes -keyout private.key -out server.csr -config lameva.cnf*

> NOTA: Existeix un paràmetre de *openssl req* que permet obviar el valor indicat a la directiva *req_extensions* de l'arxiu de configuració per tal d'utilitzar un altre valor. Es tracta del paràmetre *-reqexts extensionsCSR* Això té sentit quan a l'arxiu de configuració hi apareixen vàries seccions ("[extensionsCSR]","[altresExtensionsCSR]", etc) que contenen diferents extensions cadascuna ("subjectAltName" o altres) i es vol generar un CSR amb un d'aquests conjunts en concret.

Altres extensions diferents interessants que es poden afegir, a més de "subjectAltName", sota la mateixa secció "[extensionsCSR]" (o en una altra secció diferent: acabem de veure que la secció a utilitzar en un moment donat per generar un CSR concret es pot escollir mitjançant la directiva *req_extensions* del fitxer de configuració o bé directament mitjançant el paràmetre *-reqexts* de la comanda *openssl req ...*) són, per exemple (teniu la llista completa d'extensions possibles juntament amb una explicació de cadascuna a *man x509v3_config*):

*basicConstraints = Aquesta extensió només apareix en certificats arrel amb el valor "CA:TRUE" (sense cometes). També podria valer "CA:FALSE" (sense cometes) però llavors seria equivalent a no haver escrit aquesta extensió. Serveix per indicar precisament que el certificat en qüestió ha de ser de tipus arrel (és a dir, dissenyat per signar altres certificats).

*keyUsage = Aquesta extensió serveix per restringir els possibles usos que pot tenir el certificat en qüestió als indicats (si aquesta extensió no apareix, el certificat podrà ser emprat per qualsevol ús). Pot valer diversos valors, separats per comes. D'entre els valors més comuns (sense cometes) podem destacar: "*keyEncipherment*", "*dataEncipherment*", "*digitalSignature*" o "*nonRepudiation*" entre altres. El significat de cadascun es pot consultar a https://roll.urown.net/ca/x509.html#key-usage

*extendedKeyUsage = Aquesta extensió serveix per restringir els possibles usos que pot tenir el certificat en qüestió als indicats (si aquesta extensió no apareix, el certificat podrà ser emprat per qualsevol ús). Pot valer diversos valors, separats per comes. D'entre els valors més comuns (sense cometes) podem destacar: "*serverAuth*", "*clientAuth*", "*codeSigning*" o "*emailProtection*", entre altres.

*Crear a la vegada una clau privada P-256 i un CSR (anomenat "server.csr") signat per ella:

openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -nodes -keyout private.key -out server.csr

> NOTA: Una manera alternativa d'aconseguir el mateix seria fent-ho en dos passos: primer creant un "fitxer de paràmetres" (que anomenarem "fitxer.param"), adient pel tipus de clau el·líptica que voldrem generar, amb la comanda *openssl genpkey* ***-genparam*** *-algorithm ec -pkeyopt ec_paramgen_curve:prime256v1 -out fitxer.param* i, un cop fet això, executant llavors la comanda efectiva pròpiament dita, així: *openssl req -new -newkey* ***ec:fitxer.param*** *-nodes -keyout private.key -out server.csr*
> NOTA: Veure notes (i quadre) de l'apartat anterior per saber possibles paràmetres extra a l'hora de crear un CSR

*Crear un CSR signat amb una clau privada pròpia ja existent:

openssl req -new -key private.key -out server.csr

NOTA: Ens podem estalviar totes les preguntes interactives que aquesta comanda fa si utilitzem o bé el paràmetre *-subj* o bé un arxiu de configuració "ad-hoc", ambdues alternatives estan explicades al quadre gris anterior..
NOTA: Es pot afegir el paràmetre *-keyform {PEM|DER}* per indicar el format de la clau usada (per defecte PEM)
NOTA: Si la clau utilitzada tingués una "passphrase" integrada, es demanarà interactivament. Per evitar això, es pot escriure (suposant que el "passphrase" és "hola") el paràmetre *-passin pass:hola* També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*.
NOTA: Es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CSR per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida)
NOTA: Es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CSR Per defecte és PEM

*Comprovar el contingut d'un Certificate Signing Request (CSR):

openssl req -in server.csr -text -noout

NOTA: El paràmetre *-text* serveix per veure els camps que formen les metadades del CSR. Per saber el significat de les més importants es pot consultar https://knowledge.digicert.com/solution/SO18140.html o fer els exercicis del final.
NOTA: El paràmetre *-noout* serveix per no mostrar el contingut del CSR en sí
NOTA: Es pot afegir el paràmetre *-verify* (o escriure'l en comptes de *-text*) per simplement comprovar que el CSR estigui correctament emplenat. Només mostrarà el missatge "Verify is OK" (o "Verify is not OK")
NOTA: Es pot afegir el paràmetre *-subject* per veure les metadades introduïdes interactivament
NOTA: Es pot afegir el paràmetre *-inform {PEM|DER}* per especificar el format del CSR (per defecte és PEM)

---

*Crear a la vegada una clau privada RSA i un certificat (anomenat "server.crt") **autosignat** per ella:

openssl req -new -x509 -newkey rsa:2048 -nodes -keyout private.key -out server.crt

NOTA: És la mateixa comanda que s'usa per crear un CSR (veure l'apartat corresponent per saber el significat dels diferents paràmetres utilitzats en ella) només afegint un paràmetre extra: *-x509*. Aquesta opció és la que fa que la sortida de la comanda (fitxer indicat al paràmetre *-out*) no sigui un CSR sinó un CRT (autosignat).
NOTA: Igual que vam veure en la creació de CSRs, ens podem estalviar totes les preguntes interactives que aquesta comanda fa si utilitzem o bé el paràmetre *-subj* o bé un arxiu de configuració "ad-hoc", ambdues alternatives estan explicades als quadres grisos anteriors i següent
NOTA: Igual que vam veure en la creació de CSRs, es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CRT per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida)
NOTA: Igual que vam veure en la creació de CSRs, es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CRT Per defecte és PEM
NOTA: Es pot afegir el paràmetre *-days nº* per indicar el número de dies que el certificat tindrà validesa, més enllà dels quals caducarà. Per defecte són 30 dies.
NOTA: Es pot afegir el paràmetre *-set_serial nº* per indicar manualment un número de sèrie al certificat. Si no s'escriu, s'assignarà un número aleatori.

Aquesta comanda és molt útil per generar certificats per servidors en proves (en els que els clients hauran de confiar explícitament, ja que estarien signats per una clau a priori no reconeguda -el cas més comú d'això és quan apareix un missatge d'advertència als navegadors en el moment que s'hi vol accedir per primera vegada a un servidor HTTPS que empra aquest tipus de certificats, missatge que deixa a voluntat del visitant la decisió de confiar-ne o no-) però també per generar certificats arrel si es vol actuar com a CA de la nostra organització; en aquest darrer cas, "server.crt" seria el certificat arrel (moltes vegades renombrat a

"**ca.crt**" per a identificar-lo) que haurien de tenir els clients al seu abast -veurem als exercicis com- per tal de reconèixer a partir de llavors els certificats signats per la nostra CA sense cap missatge d'advertència, i "private.key" seria la clau privada de la CA (moltes vegades renombrada a "**ca.key**" per identificar-la) que serviria per signar tots els CSR dels diferents servidors finals de la nostra organització -a sota s'explica com-.

NOTA: En qualsevol cas, si realment es vol implementar una CA a nivell "empresarial", és convenient instal.lar i executar un software més específic com ara DogTagPKI (https://www.dogtagpki.org)

*Crear a la vegada una clau privada P-256 i un certificat (anomenat "server.crt") **autosignat** per ella:

openssl req -new -x509 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -nodes -keyout private.key -out server.crt

NOTA: Veure notes de l'apartat anterior per saber de possibles paràmetres extra a l'hora de crear un CRT

*Crear un certificat **autosignat** per una clau privada pròpia ja existent:

openssl req -new -x509 -key private.key -out server.crt

NOTA: És la mateixa comanda que s'usa per crear un CSR amb una clau pre-existent (veure l'apartat corresponent per saber el significat dels diferents paràmetres utilitzats en ella) només afegint un paràmetre extra: *-x509*. Aquesta opció és la que fa que la sortida de la comanda (fitxer indicat al paràmetre *-out*) no sigui un CSR sinó un CRT (autosignat).
NOTA: Igual que vam veure en la creació de CSRs, ens podem estalviar totes les preguntes interactives que aquesta comanda fa si utilitzem o bé el paràmetre *-subj* o bé un arxiu de configuració "ad-hoc", ambdues alternatives estan explicades als quadres grisos anteriors i següent.
NOTA: Igual que vam veure en la creació de CSRs amb clau pre-existent, es pot afegir el paràmetre *-keyform {PEM|DER}* per indicar el format de la clau usada (per defecte PEM)
NOTA: Igual que vam veure en la creació de CSRs amb clau pre-existent, si aquesta clau tingués una "passphrase" integrada, es demanarà interactivament. Per evitar això, es pot escriure (suposant que el "passphrase" és "hola") el paràmetre *-passin pass:hola* També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*
NOTA: Igual que vam veure en la creació de CSRs, es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CRT per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida)
NOTA: Igual que vam veure en la creació de CSRs, es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CRT Per defecte és PEM
NOTA: Es pot afegir el paràmetre *-days nº* per indicar el número de dies que el certificat tindrà validesa, més enllà dels quals caducarà. Per defecte són 30 dies.
NOTA: Es pot afegir el paràmetre *-set_serial nº* per indicar manualment un número de sèrie al certificat. Si no s'escriu, s'assignarà un número aleatori.

Per tal d'afegir extensions al CRT generat amb la comanda *openssl req -new -x509 ...*, es fa de la mateixa manera que vam veure en generar CSRs amb la comanda *openssl req -new ...* però amb una diferència només: la línia "*req_extensions*" de l'arxiu "lameva.cnf" deixa de tenir sentit i cal ara substituïr-la per una línia anomenada "*x509_extensions*" (amb el mateix valor, això sí: el nom de la secció que agruparà, dins de l'arxiu, les extensions a incorporar). Per tant, un cop gravat el fitxer "lameva.cnf" amb un contingut similar al mostrat a continuació, podrem executar la mateixa comanda que faríem servir per generar el CRT normalment però afegint el paràmetre (ja conegut) *-config /ruta/lameva.cnf* ; és a dir, per exemple, així: *openssl req -new -x509 -key private.key -out server.crt* ***-config /ruta/lameva.cnf***

```
[req]
distinguished_name = dnCSR
x509_extensions = extensionsCSR
prompt = no
[dnCSR]
C = US
ST = New York
L = Brooklyn
```

```
O = Company
OU = Department
CN = www.example.com
emailAddress = admin@example.com
[extensionsCSR]
subjectAltName = @seccioParticular
[seccioParticular]
DNS.1=www.example.com
DNS.2=other.example.org
```

NOTA: Existeix un paràmetre de *openssl req -x509* que permet obviar el valor indicat a la directiva *x509_extensions* de l'arxiu de configuració per tal d'utilitzar un altre valor. Es tracta del paràmetre *-extensions extensionsCSR* Això té sentit quan a l'arxiu de configuració hi apareixen vàries seccions ("[extensionsCSR]","[altresExtensionsCSR]", etc) que contenen diferents extensions cadascuna ("subjectAltName" o altres) i es vol generar un CRT amb un d'aquests conjunts en concret.

*Crear un certificat **autosignat** a partir d'un CSR i una clau privada propis ja existents:

openssl x509 -req -in ca.csr -out ca.crt -signkey ca.key

NOTA: El paràmetre *-req* serveix per indicar que el fitxer indicar al paràmetre *-in* és un CSR (ja que per defecte la comanda *openssl x509* ... espera un CRT com entrada)
NOTA: Sovint aquesta comanda es fa servir per generar el certificat arrel d'una CA (que aquí anomenarem "CA.crt").El CSR del qual es parteix llavors no ha de tenir cap FQDN com a valor de CN sinó qualsevol altra cosa
NOTA: Aquesta comanda no pregunta res interactivament perquè totes les dades les obté del CSR proporcionat
NOTA: Igual que vam veure a l'apartat anterior, es pot afegir el paràmetre *-keyform {PEM|DER}* per indicar el format de la clau usada (per defecte PEM)
NOTA: Igual que vam veure a l'apartat anterior, si aquesta clau tingués una "passphrase" integrada, es demanarà interactivament. Per evitar això, es pot escriure (suposant que el "passphrase" és "hola") el paràmetre *-passin pass:hola* També es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*
NOTA: Igual que vam veure en la creació de CSRs i CRTs, es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CRT per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida).
NOTA: Igual que vam veure en la creació de CSRs i CRTs, es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CRT Per defecte és PEM
NOTA: Com sempre que volguem crear un CRT, es pot afegir el paràmetre *-days nº* per indicar el número de dies que el certificat tindrà validesa, més enllà dels quals caducarà. Per defecte són 30 dies.
NOTA: Com sempre que volguem crear un CRT, es pot afegir el paràmetre *-set_serial nº* per indicar manualment un número de sèrie al certificat. Si no s'escriu, s'assignarà un número aleatori.

*Crear un certificat **signat** a partir d'un CSR d'altri i un CRT i clau privada pròpia (fer de CA):

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -out server.crt -set_serial nº

NOTA: El paràmetre *-set_serial nº* serveix per indicar manualment un número de sèrie al certificat.
NOTA: Ens podem estalviar d'escriure el paràmetre *-set_serial nº* cada cop que volguem crear un certificat si escrivim un sol cop (la primera vegada que creem un primer certificat) el paràmetre *-CAcreateserial* . Aquest paràmetre assigna un número aleatori com a número de sèrie del certificat en qüestió i guarda aquest número en un fitxer anomenat "ca.srl" (ubicat a la mateixa carpeta on s'hagi executat la comanda *openssl*). Per crear els següents certificats a partir d'aquest primer ja no caldrà indicar cap paràmetre específic (ni *-set_serial nº* ni *-Cacreateserial*, és a dir, que es pot escriure simplement això: *openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -out server.crt*) perquè OpenSSL usarà el valor actual guardat a "ca.srl" per augmentar-lo automàticament i assignar aquest nou valor augmentat al nou certificat (guardant llavors aquest nou valor dins de "ca.srl" -sobreescrivint el valor anterior-, per tenir-lo disponible per la propera vegada que volem crear un nou certificat).
NOTA: Es pot indicar un fitxer "ca.srl" explícitament per tal de fer-lo servir a l'hora de crear nous certificats amb el paràmetre *-CAserial /ruta/unaltrefitxer.csl*
NOTA: Igual que vam veure a l'apartat anterior, es pot afegir el paràmetre *-keyform {PEM|DER}* per indicar el format de la clau usada (per defecte PEM)
NOTA: Igual que vam veure a l'apartat anterior, si aquesta clau tingués una "passphrase" integrada, es demanarà interactivament. Per evitar això, es pot escriure (suposant que el "passphrase" és "hola") el paràmetre *-passin*

pass:hola También es pot indicar així: *-passin env:VAR* si la variable d'entorn VAR especificada té com a valor la "passphrase" adient, o també així: *-passin file:/ruta/fitxer.txt* si el fitxer especificat (el qual podria estar al seu torn xifrat amb algun algoritme simètric...) té com a primera línia el valor de la "passphrase" adient, o també així: *-passin stdin* si la "passphrase" es rebrà a través d'una canonada. Per saber més opcions consulta l'apartat "Pass Phrase Options" de *man openssl*

NOTA: Igual que vam veure en la creació de CSRs i CRTs, es pot afegir un paràmetre "-YYY", el qual representarà el nom de l'algoritme "hash" escollit (precedit per un guió) que s'aplicarà al CRT per tal de garantir-ne la seva integritat. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com per exemple "**sha256**". De totes formes cal saber que, tot i que no s'indiqui el paràmetre "-YYY", sempre s'hi aplicarà un "hash" per defecte: l'indicat a l'arxiu de configuració general de OpenSSL (del qual en parlarem de seguida).

NOTA: Igual que vam veure en la creació de CSRs i CRTs, es pot afegir el paràmetre *-outform {PEM|DER}* per especificar el format del CRT Per defecte és PEM

NOTA: Com sempre que volguem crear un CRT, es pot afegir el paràmetre *-days nº* per indicar el número de dies que el certificat tindrà validesa, més enllà dels quals caducarà. Per defecte són 30 dies.

NOTA: Una altra comanda alternativa que permet obtenir el mateix resultat és *openssl ca ...* però no l'estudiarem

Si es volen afegir extensions extra al CRT generat més enllà de les demanades al CSR utilitzat, es pot escriure un fitxer específic (que anomenarem "mesExt.cnf") on només apareguin llistades les extensions desitjades (per exemple, així)...

```
basicContraints = CA:FALSE
subjectAltName = @seccioParticular
[seccioParticular]
DNS.1=www.example.com
DNS.2=other.example.org
```

...i llavors executar la mateixa comanda que faríem servir però afegint el paràmetre *-extfile /ruta/mesExt.cnf* ; és a dir, per exemple, així: *openssl x509 -req -in ca.csr -out ca.crt -signkey ca.key* ***-extfile mesExt.cnf***

NOTA: Dins de l'arxiu "mesExt.cnf" es poden agrupar conjunts d'extensions sota seccions. Si es fa així, per indicar quina secció concreta es vol fer servir a l'hora de generar el certificat cal afegir, a més del paràmetre *-extfile*, el paràmetre *-extensions nomSeccio* (si es deixen extensions fora de qualsevol secció, es consideraran dins d'una secció anomenada "default")

*Comprovar el contingut d'un certificat: *openssl x509 -in server.crt -text -noout*

NOTA: El paràmetre *-text* mostra els camps que formen les metadades del CRT. Per saber el significat de les més importants es pot consultar https://knowledge.digicert.com/solution/SO18140.html o fer els exercicis del final.

NOTA: El paràmetre *-noout* serveix per no mostrar el contingut del certificat en sí

NOTA: Es pot afegir el paràmetre *-subject* per veure les metadades introduïdes interactivament

NOTA: Es pot afegir el paràmetre *-inform {PEM|DER}* per especificar el format del CRT (per defecte és PEM)

---

*Obtenir el "fingerprint" d'un certificat: *openssl x509 -in server.crt -YYY -fingerprint -noout*

NOTA: El paràmetre "-YYY" representa el nom de l'algoritme "hash" escollit (precedit per un guió) per obtenir el fingerprint. Aquest nom pot ser un de la llista mostrada per *openssl list -digest-commands* , com p. ex."**sha256**"

*Comprovar el DN d'un certificat: *openssl x509 -in server.crt -subject -noout*

*Comprovar el número de sèrie d'un certificat: *openssl x509 -in server.crt -serial -noout*

*Comprovar la data de caducitat d'un certificat: *openssl x509 -in server.crt -dates -noout*

NOTA: Es mostraran les dates "notBefore" i "notAfter". La que ens interessa conèixer per saber si un certificat ha expirat o no (i, per tant, si continua sent vàlid o no) és la de "notAfter".

*Saber quina/es CA/s han signat un certificat: *openssl x509 -in server.crt -issuer -noout*

NOTA: També es pot usar la comanda *openssl verify [-CAfile ca.crt] server.crt* , la qual mostra amb un "OK" si el certificat indicat ha sigut signat per una CA determinada (identificada pel certificat arrel indicat, si aquesta no fos ja reconeguda per defecte pel sistema)

*Saber si el certificat és de tipus servidor, client i/o CA: *openssl x509 -in server.crt -purpose -noout*

NOTA: També es pot usar la comanda *openssl verify -purpose sslserver -CAfile ca.crt server.crt* , la qual mostra amb un "OK" si el certificat indicat serveix com a certificat de servidor de tercers

*Convertir un certificat PEM a DER: *openssl x509 -in server.pem -out server.der -outform der*

NOTA: Recordar que el format DER és binari (en format ASN.1) i el format PEM és de tipus text (conté la representació en Base64 del certificat DER en sí, precedit de la línia ---BEGIN CERTIFICATE--- i finalitzat per la línia ---END CERTIFICATE---) En un mateix fitxer PEM poden haver més d'un certificat (separats entre sí per les línies descrites) però no és habitual

*Convertir un certificat DER a PEM: *openssl x509 -in server.der -out server.pem -inform der*

NOTA: If you need to use a cert with the java application (or any other who accept only PKCS#7 format) or with a Microsoft application (or any other who accept only PKCS#12 -PFX-), you can use similar commands to convert from/into this format

Si en algun moment volguéssim comprovar si la clau pública inclosa dins d'un certificat és la mateixa que la que està dins d'un eventual CSR o és la mateixa que es va generar a partir d'una determinada clau privada, podríem comparar la clau pública extreta de cadascun d'aquests artefactes (certificat, CSR, clau privada) és la mateixa executant les comandes següents (on a cada clau pública obtinguda se li aplica un hash SHA256 per facilitar-ne la comparació -això darrer no és imprescindible però sí més còmode-):

openssl x509 -in server.crt -pubkey -noout | openssl sha256
openssl req -in server.csr -pubkey -noout | openssl sha256
openssl pkey -in private.key -pubout | openssl sha256

---

*Conectar a un servidor TLS (HTTP**S**, SMTP**S**, LDAP**S**,...)

openssl s_client -connect www.marca.com:443

NOTA: Per saber si una versió concreta del protocol TLS és acceptada pel servidor, es pot forçar a realitzar la connexió amb una versió concreta afegint el paràmetre respectiu *-ssl3*, *-tls1*, *-tls1_1*, *-tls1_2*, *-tls1_3* (o *-no_ssl3*, *-no_tls1*, *-no_tls1_1*, etc). Si aquesta versió és acceptada pel servidor, s'obtindrà la resposta "CONNECTED"; si no, un "handshake failure." (a les línies anteriors es pot comprovar el valor "Protocol" i "Cipher" per esbrinar realment amb quina versió de TLS i quina ciphersuite està contestant el servidor per establir la connexió.
NOTA: Per saber si una "ciphersuite" concreta és acceptada pel servidor, es pot forçar a realitzar la connexió amb una "ciphersuite" determinada, així *-cipher 'ECDHE-ECDSA-AES256-SHA'* (si s'usa TLS1.2 o inferior) o *-ciphersuites 'TLS_AES_256_GCM_SHA384'* (si s'usa TLS1.3 o superior; es poden provar vàries "ciphersuites" en ordre de preferència si s'escriuen una darrera de l'altra separades per ":"). Si és acceptada, s'obtindrà la resposta "CONNECTED"; si no, un “handshake failure.”. Sobre el concepte de "ciphersuite" en parlarem més endavant.
NOTA: Si el certificat del servidor fos signat per una CA no reconeguda pel nostre sistema, caldrà indicar "a mà" el certificat arrel adient "a mà" amb el paràmetre *-CAfile /ruta/ca.crt*
NOTA: També és possible utilitzar la sintaxis *openssl s_client -host www.marca.com -port 443*
NOTA: A la secció "Certificate chain" es poden veure els detalls del "subject" (línies que comencen per "s") i de l'"issuer" (línies que comencen per "i") de tots els certificats de la cadena (les primeres línies es corresponen al certificat final i les darreres a l'arrel). Si escrivim el paràmetre *-showcerts* podrem veure, a més, tots els certificats pròpiament dits que formen aquesta cadena. També és interessant el paràmetre *-tlsextdebug* per obtenir informació sobre les extensions detectades al certificat del servidor (el darrer de la cadena)
NOTA: Es pot verificar si el certificat ofert per un servidor remot cobreix un determinat nom DNS a més del canònic (això és útil per comprovar si el certificat és multidomini realment) amb la comanda *openssl s_client -verify_hostname www.example.com -connect example.com:443*
NOTA: Entre altres informacions que ens mostra la comanda, podem veure un "Session ID" i un "Session Ticket", els quals permeten reempredre sessions sense que el servidor hagi de mantenir l'estat. En aquest sentit, és interessant el paràmetre *-reconnect* , el qual crea una connexió i la reutilitza cinc vegades més, mostrant per pantalla la informació pertinent de la sessió inicial i després la dels restabliments d'aquesta.
NOTA: En el cas que el client utilitzi un certificat i clau propis per autenticar-se contra el servidor, caldrà afegir els corresponents paràmetre *-cert* i *-key* (veieu l'enunciat de l'exercici nº10 per més informació)

Aquesta comanda funciona com un client *ncat –ssl* ... però més enllà del seu possible ús com a client de consola (bé interactiu bé agafant l'entrada d'una canonada), és útil sobre tot per comprovar els detalls de la connexió TLS (sobre tot si s'usa el paràmetre *-showcerts* o també *-debug*) perquè mostra els detalls de tots els certificats involucrats en cadena durant la connexió TLS fins que aquesta s'estableix finalment (event que s'indica amb la impressió de tres guions ("---") per pantalla), moment en el qual ja es pot "començar a treballar" (fent servir el protocol superior que hi hagi implementat al servidor: si aquest fos HTTP, per exemple, es podria escriure directament una petició GET, etc).

NOTA: Sota l'apartat "Diagnostics" de la pàgina *man openssl-verify* es troben llistats els diferents codis de retorn possibles en l'establiment d'una connexió TLS i una explicació del què significa cadascú (molt útil si hi ha hagut algun error i no sabem per què)

NOTA: D'altra banda, es poden fer diferents proves de velocitat en les connexions TLS amb un servidor remot amb les comandes *openssl s_time -connect www.marca.com:443 -new* (per una nova connexió) o *openssl s_time -connect www.marca.com:443 -reuse* (per una connexió ja establerta)

*Posar en marxa un servidor TLS "a seques":

openssl s_server -key private.key -cert server.crt [-port nº] [-quiet]

NOTA: Per defecte, si no s'indica, el port a usar serà el 4433. Una altra manera d'indicar el port (i la IP concreta a la qual estarà associada és utilitzant el paràmetre *-accept IP:nºport*
NOTA: Es pot afegir el paràmetre *-WWW* per a simular un servidor web senzill que pot carregar pàgines HTML.
NOTA: Per forçar als clients a utilitzar una versió concreta del protocol TLS (per exemple, TLS 1.3) es pot utilitzar el paràmetre *-tls1_3* (o paràmetres similars). Si no s'indica cap, el servidor negociarà amb el client la versió més moderna que sigui entesa per ambdós extrems. Un cop indicada la versió del protocol, es pot restringir fins i tot la/es ciphersuite/s a utilitzar mitjançant el paràmetre *-ciphersuites una:unaAltra:...* (noteu que quan un client envia la seva llista de ciphersuites suportades, el primer valor inclòs en aquesta llista que apareixi com a valor d'aquest paràmetre *-ciphersuites* serà l'utilitzat; és a dir, que l'ordre el marca la llista del client, no pas com s'hagin escrit al paràmetre *-ciphersuites*)
NOTA: Per saber el significat i utilitat dels paràmetre *-verify nº* i *-Verify nº* consulteu l'enunciat de l'exercici 10. En aquest sentit, si el client que s'hi connecta s'autentiqués mitjançant un certificat de client i aquest fos signat per una CA no reconeguda pel nostre servidor, caldrà indicar "a mà" el certificat arrel adient "a mà" amb el paràmetre *-CAfile /ruta/ca.crt*

Aquesta comanda funciona com un servidor *ncat --ssl -l -p* ... encara que més enllà del seu possible ús com a servidor interactiu de consola, és útil sobre tot per comprovar els detalls de la connexió TLS.

---

*Saber la versió d'OpenSSL instal.lada al sistema: *openssl version [-a]*

*Saber les "ciphersuites" TLS que el nostre OpenSSL pot gestionar: *openssl ciphers -s [-v]*

NOTA: El paràmetre *-s* serveix per efectivament mostrar només les "ciphersuites" suportades pel sistema actual concret, no totes les teòricament existents (que és el que es veu sense aquest paràmetre).
NOTA: El paràmetre *-v* és el mode verbós: mostra més dades sobre cada "ciphersuite" mostrada
NOTA: Es pot restringir la sortida de la comanda anterior segons el tipus de "ciphersuite" desitjada si s'indica alguna característica concreta com a paràmetre. Per exemple, es pot indicar *-tls1_3* per mostrar només les "ciphersuites" reconegudes usades en el protocol TLS1.3 (les úniques recomanables).

*Comprovar la velocitat de diferents algoritmes de xifrat al nostre sistema: *openssl speed*

NOTA: There are three relevant parts to the output. The first part consists of the OpenSSL version number and compile-time configuration. This information is useful if you’re testing several different versions of OpenSSL with varying compile-time options. The second part contains symmetric cryptography benchmarks (i.e., hash functions and private cryptography). Finally, the third part contains the asymmetric (public) cryptography benchmarks
NOTA: Es pot fer que els tests es realitzin en vàries CPU a la vegada amb el paràmetre *-multi nº*
NOTA: Es pot restringir la prova a un determinat algoritme (de xifrat o de "hash"), el nom del qual s'haurà d'indicar com a darrer paràmetre. Per saber els noms possibles es pot escriure *openssl list -cipher-commands* o *openssl list -digest-commands*

EXERCICIS:

1.-a) Codifica una imatge PNG qualsevol que tinguis disponible al sistema en Base64. Això ho pots fer executant la comanda *openssl enc -e -base64 -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)?

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 "file magic" del fitxer) i genera la imatge corresponent a partir d'aquest fitxer modificat executant la comanda *openssl enc -d -base64 -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*?

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?

2.-a) Crea un arxiu de tipus "tar.gz" que inclogui la carpeta "~/Imatges" (amb tot el seu contingut). Això ho pots fer mitjançant la comanda *tar -czf Fotos.tar.gz ~/Imatges* (en aquest exemple l'arxiu "tar.gz" s'anomenarà "Fotos.tar.gz")

aII) Comprova 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 -iter 2 -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 -iter 2 -pass pass:1234 -in Fotos.tar.gz.enc -out FotosFinal.tar.gz*

cII) 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 -iter 2 -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)
>
> NOTA: Es pot dexifrar un arxiu "tar.gz" i desempaquetar-lo a la vegada executant *openssl enc -d -aes-256-cbc -iter 2 -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)
>
> NOTA: 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". En aquest sentit, és molt interessant l'article https://myexperiments.io/finding-cipher-algorithm-encrypted-file.html

d) 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 -t 4 -l 5 -m 5 -s "0123456789abcdef" -b "00" -c aes256 encrypted.file* ¿I aquesta: *bruteforce-salted-openssl -t 4 -v 30 -f dictionary.txt -w state.txt -c aes256 encrypted.file* ?

3.-Executa aquesta línia en un terminal (la qual farà bloquejar el terminal) i espera't uns segons fins pulsar CTRL+C: *while [true] ; do openssl rand -hex 1 | openssl dgst -md5 ; done >> a.txt* A partir de saber què fan les comandes *openssl rand* i *openssl dgst* anteriors, explica el significat del resultat mostrat en executar *sort a.txt | uniq -c*

4.-a) Crea una clau privada RSA de 2048 bits sense "passphrase" i anomenada "ca.key". Això ho pots aconseguir executant la comanda *openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:2048 -out ca.key*

aII) A partir del que veus en executar *cat ca.key* (o també *openssl pkey -in ca.key*) dedueix si el format de la clau generada és PEM o DER.

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

b) Genera, a partir de la clau privada que ja tens, la corresponent clau pública. Això ho pots aconseguir executant la comanda *openssl pkey -in ca.key -pubout -out public.key* (aquí anomenarem "public.key" a aquesta clau pública)

bII) A partir del que veus en executar *cat public.key* (o també *openssl pkey -pubin -in public.key*) dedueix 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 amb la comanda *openssl pkeyutl -encrypt -in Fotos.tar.gz -out Fotos.tar.gz.enc -pubin -inkey public.key*. Un cop hagi obtingut el fitxer xifrat (que suposarem que es diu "Fotos.tar.gz.enc"), te l'haurà d'enviar d'alguna manera (mail, ncat, scp, ...és igual).

> NOTA: Because of the nature of the RSA algorithm, a single encryption process can only encrypt input data that is smaller than the RSA key size. In other words, the size (number of bytes) of the input data should be smaller than the size (number bytes) of the RSA key.

cALTERNATIU) Si el teu company ha obtingut un error del tipus "data too large for key size" en fer l'apartat anterior (llegeix la nota corresponent), pot solventar aquest problema seguint els passos següents:

> *Creant una clau simètrica (és la única manera de xifrar fitxers grans si no es volen fer servir algoritmes asimètrics de fluxe, més complicats). Una manera ràpida de fer-ho és amb aquesta comanda: *openssl rand -hex -out key.bin 32* (anomenarem "key.bin" a aquesta clau)
>
> *Xifrant el fitxer gran amb aquesta clau, igual que vam fer a l'exercici 2 però no utilitzant cap "passphrase" (que l'unic que serveix és per generar la clau corresponent com hem fet a mà al pas anterior) sinó directament la clau simètrica que hem generat explícitament al pas anterior, així: *openssl enc -e -aes-256-cbc -iter 2 -in Fotos.tar.gz -out Fotos.tar.gz.enc -pass file:./key.bin*
>
> *Xifrant "key.bin" amb la clau pública "public.key" del destinatari, així: *openssl pkeyutl -encrypt -in key.bin -out key.bin.enc -pubin -inkey public.key* (a més, es recomana destruir completament l'arxiu "key.bin" per a què ningú el pugui fer servir, així per exemple *shred -u key.bin*)
>
> *Enviant al destinatari tant el fitxer "Fotos.tar.gz.enc" com "key.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 ca.key* ¿Què obtens?

dALTERNATIU) Desxifra l'arxiu "key.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 key.bin.enc -out key.bin -inkey ca.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 -iter 2 -in Fotos.tar.gz.enc -out Fotos.tar.gz -pass file:./key.bin* ¿Què obtens?

5.-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?

b) Torna a repetir l'apartat anterior 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 ca.key -out a.txt.dgst a.txt* ¿El contingut del fitxer "a.txt.dgst" és diferent de l'homònim creat a l'apartat anterior?

> NOTA: Per obtenir una versió en mode text de "a.txt.dgst" (el seu contingut és binari) es pot codificar aquest a Base64.
> NOTA: Imagina que "a.txt" fos una clau pública...signant-la estaríem garantint la seva autenticitat -que és el que fan, en essència, els certificats-; si "a.txt" fos, en canvi, un document qualsevol, signant-lo estaríem garantint la seva integritat.

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?

6.-a) Crea un certificat autosignat, que s'anomeni "ca.crt" i que tingui un any de duració, a partir de la clau privada generada a l'apartat a) de l'exercici 4 (o d'una altra qualsevol, fins i tot nova, és igual). Aquest CRT serà una mica especial perquè no serà un certificat de servidor estàndar sinó que funcionarà com certificat arrel per poder actuar com una CA privada i així generar certificats pels nostres diferents servidors; per tant, a més d'incloure al certificat la informació típica sobre país, província, ciutat, empresa i departament, en el "Common Name" s'hi pot indicar un nom qualsevol, com ara "My CA". Això ho pots aconseguir (de forma no interactiva) executant la comanda *openssl req -new -x509 -days 365 -key ca.key -out ca.crt -subj "/C=AD/ST=Ordino/L=Ordino/O=Spy INC./OU=Hackers/CN=My CA"*

> NOTA: No cal dir que "ca.key" no hauria de sortir mai de la màquina: si surt tot l'esquema de seguretat estarà compromés
> NOTA: En comptes de generar un certificat arrel per després signar els certificats dels nostres servidors, es podria igualment generar un certificat autosignat diferent per cada servidor que administréssim (igual que s'acaba de fer a l'apartat anterior). No obstant, tenint una entitat certificadora centralitzada és molt més professional perquè els clients només hauran de posseir en un únic certificat arrel ("ca.crt") per poder confiar en els diferents certificats de servidor en comptes d'haver de posseir diferents certificats autosignats, un per cadascun d'aquests servidors.

b) Ara suposarem que estem en una màquina que farà de servidor HTTPS i, per tant, necessitarà la seva pròpia clau privada i el seu certificat corresponent. Pots obrir una altra màquina virtual diferent de la què farà de CA per fer-ho més realista però no és necessari si no vols. En qualsevol cas, en aquest hipotètic servidor HTTPS cal que executis primer la comanda següent per crear la seva pròpia clau privada (la qual ha de ser hiperprotegida!), per exemple així: *openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:2048 -out serverHttps.key* i tot seguit, cal que executis la comanda següent per crear un CSR, el qual s'haurà d'enviar d'alguna manera (via mail, ncat, scp...és igual) a la màquina que fa de CA: *openssl req -new -key serverHttps.key -out serverHttps.csr -subj "/C=AD/ST=Andorra/L=Andorra/O=Hostings INC./OU=DevOps/CN=www.elmeunom.com"* (on "www.*elmeunom*.com" -com per exemple, "www.oscar.com"-, representa el nom DNS del servidors HTTPS; tingueu en compte que encara que ara mateix aquest nom no estigui definit en cap servidor DNS, és important que sigui aquest pels propers exercicis).

c) Un cop la màquina que fa de CA tingui al seu poder l'arxiu "serverHttps.csr", caldrà que allí executis la següent comanda per tal de generar el certificat del servidor HTTPS corresponent, amb una duració d'un any (i se suposa que se li retornarà per a què ell el pugui fer servir a la seva configuració): *openssl x509 -req -days 365 -in serverHttps.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out serverHttps.crt*

> NOTA: Tal com s'ha explicat a la teoria, la comanda anterior generarà, a més del fitxer "serverHttps.crt", un fitxer anomenat "ca.srl" incloent el número de sèrie aleatori utilitzat per generar el primer. És molt important no perdre aquest fitxer!!

7.-a) Aprofita la clau privada que vas crear a l'apartat b) de l'exercici anterior per tornar a generar un altre CSR que tingui el mateix DN però que ara inclogui a més l'extensió "subjectAltName" amb els valors "www.elmeunom.com", "elmeunom.com", "*.elmeunom.com", "*.elmeunom.org" i "*.cocacola.com". Això ho pots fer executant la comanda *openssl req -new -key serverHttps.key -out serverHttps2.csr -config serverHttps2.cnf* tenint en compte que l'arxiu "serverHttps2.cnf" ha de tenir un contingut com el següent:

[req]
distinguished_name = dnCSR
req_extensions = extensionsCSR
prompt = no
[dnCSR]
C = AD
ST = Andorra
L = Andorra
O = Hostings INC.
OU = DevOps
CN = www.elmeunom.com
emailAddress = admin@elmeunom.com
[extensionsCSR]
subjectAltName =
DNS:www.elmeunom.com,DNS:elmeunom.com,DNS:.elmeunom.com,DNS:*.elmeunom.org,DNS:*.cocacola.com*

b) Actua ara com a CA; és a dir: signa el CSR generat a l'apartat anterior (que inclou l'extensió "subjectAltName") executant la comanda *openssl x509 -req -in serverHttps2.csr -CA ca.crt -CAkey ca.key -out serverHttps2.crt -set_serial 12345* ¿Per a quins dominis aquest certificat serà vàlid?

c) Crea ara un certificat autosignat (que també incorpori l'extensió "subjectAltName") anomenat "serverHttps3.crt" executant la comanda *openssl req -new -x509 -key ca.key -out serverHttps3.crt -config serverHttps3.cnf* (on l'arxiu "serverHttps3.cnf" haurà de tenir el següent contingut):

[req]
distinguished_name = dnCSR
x509_extensions = extensionsCSR
prompt = no
[dnCSR]
C = AD
ST = Andorra
L = Andorra
O = Hostings INC.
OU = DevOps
CN = www.elmeunom.com
emailAddress = admin@elmeunom.com
[extensionsCSR]
subjectAltName =
DNS:www.elmeunom.com,DNS:elmeunom.com,DNS:.elmeunom.com,DNS:*.elmeunom.org,DNS:*.cocacola.com*

NOTA: Al següent exercici podràs comprovar si has realitzat correctament els tres apartats anteriors

8.- a) Executa la comanda *openssl x509 -in serverHttps.crt -text -noout* (on "serverHttps.crt" representa el certificat creat a l'apartat *c)* de l'exercici 6) i localitza el valor dels següents camps entre els que el formen:

-Número de sèrie del certificat
-Algoritme emprat per calcular el "hash" del certificat (valor indicat com a "Signature Algorithm") i el propi "hash" (valor indicat al final de tot)
NOTA: Si no es va indicar cap algoritme "hash" en concret a la comanda *openssl req -new ...* , el camp "Signature Algorithm" mostrarà l'utilitzat per defecte gràcies a la configuració establerta a l'arxiu openssl.cnf
-DN de la CA signant del certificat
-Data d'inici i de final de validesa del certificat
-DN del propi certificat
-Algoritme emprat per generar la clau pública inclosa dins del certificat (com que aquesta deriva de la clau privada emprada, serà el mateix algoritme que l'utilitzat en la creació d'aquesta)

b) Executa la comanda *openssl x509 -in ca.crt -text -noout* (on "ca.crt" representa el certificat autosignat creat a l'apartat *a)* de l'exercici 6) i confirma que:

-El DN de la CA signant del certificat i el DN del propi certificat tinguin el mateix valor
-Apareguin, sota l'apartat "X509v3 Extensions" (que al certificat inspeccionat a l'apartat anterior no hi era), les extensions "SKI", "AKI" i "CA:TRUE" . Consulta la pàgina *man x509v3_config* per esbrinar per a què serveix cadascuna

c) Executa la comanda *openssl req -in serverHttps.csr -text -noout* (on "serverHttps.csr" representa el CSR creat a l'apartat *b)* de l'exercici 6) i localitza el valor dels següents camps entre els que el formen:

-DN del CSR
-Algoritme emprat per generar la clau pública inclosa dins del CSR (com que aquesta deriva de la clau privada emprada, serà el mateix algoritme que l'utilitzat en la creació d'aquesta)
-Algoritme emprat per calcular el "hash" del CSR (valor indicat com a "Signature Algorithm") i el propi "hash" (valor indicat al final de tot)

NOTA: Si no es va indicar cap algoritme "hash" en concret a la comanda *openssl req ...* , el camp "Signature Algorithm" mostrarà l'utilitzat per defecte gràcies a la configuració establerta a l'arxiu openssl.cnf

d) Executa la comanda *openssl req -in serverHttps2.csr -text -noout* (on "serverHttps2.csr" representa el CSR creat a l'apartat *a)* de l'exercici 7) i confirma que apareix una secció dins d'aquest CSR (que no apareixia en "serverHttp.csr") anomenada "Request Extensions" incloent l'extensió "X509v3 Subject Alternative Name" amb el valor adient.

e) Executa la comanda *openssl x509 -in serverHttps2.crt -text -noout* (on "serverHttps2.crt" representa el certificat creat a l'apartat *b)* de l'exercici 7) i confirma que apareix una secció dins d'aquest CRT (que no apareixia en "serverHttps.crt") anomenada "X509v3 extensions" incloent l'extensió "X509v3 Subject Alternative Name" amb el valor adient.

NOTA: Hauries de veure l'extensió però actualment existeix el següent bug: "Extensions in certificate requests are not transferred to certificates" (https://www.openssl.org/docs/man1.1.0/man1/x509.html#BUGS) que fa que "serverhttp2.crt" no inclogui extensions.

f) Executa la comanda *openssl x509 -in serverHttps3.crt -text -noout* (on "serverHttps3.crt" representa el certificat creat a l'apartat *c)* de l'exercici 7) i confirma que apareix -ara sí- que apareix una secció dins d'aquest CRT anomenada "X509v3 extensions" incloent l'extensió "X509v3 Subject Alternative Name" amb el valor adient.

9.-a) Executa la comanda següent (on el certificat "serverHttps.crt" i la clau privada "serverHttps.key" els vas haver de crear a l'apartat *c)* i *b)* de l'exercici 6, respectivament) : *openssl s_server -tls1_3 -key serverHttps.key -cert serverHttps.crt* . A continuació (en un altre terminal) observa la sortida de la comanda *sudo ss -tlnp* i digues en quin port i per a quines IPs rebrà peticions TLS el servidor que acabes de posar en marxa.

aII) Connecta al servidor anterior executant (en un terminal d'on el servidor està en marxa) la comanda *openssl s_client -connect 127.0.0.1:4433* i confirma que la connexió s'ha realitzat correctament enviant la paraula "Hola" al servidor. A partir d'aquí, troba a la pantalla del client els següents valors informatius (mostrats durant el procés d'establiment de la connexió TLS) i digues què signifiquen:

-La frase *"verify error:num=20:unable to get local issuer certificate*" (i com a conseqüència, la frase "*verify error:num=21:unable to verify the first certificate*"
-Les línies que comencen per "s:" i "i:" sota l'apartat "Certificate chain"
-El contingut Base64 mostrat entre les línies ---*BEGIN CERTIFICATE*--- i ---*END CERTIFICATE*---
-El valor que segueix a la paraula "New," al principi d'una línia

NOTA: El significat del valor que segueix a "Cipher is" en aquesta mateixa línia l'estudiarem més endavant
En qualsevol cas, es pot comprovar que aquest valor canvia si afegim a la comanda *openssl s_client ...* anterior els paràmetres *-tls1_3 -ciphersuites TLS_CHACHA20_POLY1305_SHA256* , per exemple
NOTA: Tant el valor que segueix a "New," com el que segueix a "Cipher is" els tornem a trobar en línies següents, sota la secció "SSL-Session", acompanyant a les opcions "Protocol" i "Cipher", respectivament

-El valor que segueix a la paraula "Server Public Key is" al principi d'una línia
-El valor "Session ID", "TLS Session Ticket" i "Timeout" presents sota la secció "SSL-Session" (consulta les notes de teoria sobre la comanda *openssl s_client* per saber-ho)

aIII) Desconnecta el client del servidor (pulsant CTRL+C) i torna-hi a connectar però ara indicant el certificat arrel amb el qual es va signar el seu certificat (en el nostre cas, "ca.crt"). Això ho pots fer amb la comanda *openssl s_client -connect 127.0.0.1:4433 -CAfile ca.crt* ¿Quina diferència veus en els valors informatius que ara apareixen durant el procés d'establiment d'aquesta nova connexió TLS respecte els que vas veure a l'apartat anterior (fixa't sobre tot en les línies que comencen per "Verification error" (o "Verification: OK") i "Verify return code")?

b) Executa la comanda *echo -en "GET / HTTP/1.1\r\nHost:www.wikipedia.org\r\n\r\n" | openssl s_client -connect www.wikipedia.org:443* i fixa't en les línies de la secció "Certificate chain". ¿Quina és la CA arrel? ¿Aquesta CA, a quina altra CA intermitja signa? ¿Aquesta CA intermitja quin tipus de certificat de servidor signa ("wildcard", SAN,...)? ¿En quina ciutat està legalment ubicada la fundació Wikimedia?

bII) ¿Quina informació extra veus si afegeixes a la comanda *openssl s_client* anterior el paràmetre *-showcerts* ?

c) Digues per a què serveix el següent script:

```
#!/bin/bash
for CERT in \
  www.marca.com:443 \
  www.elmundo.es:443
do
  echo |\
  openssl s_client -connect ${CERT} 2>/dev/null |\
  sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' |\
  openssl x509 -noout -subject -dates
done
```

> NOTA: Les contrabarres "\" serveixen per indicar que la comanda segueix (és per no interpretar el salt de línia com a final)
> NOTA: La comanda *echo* entubada a *openssl s_client* és un truc per eliminar la interactivitat d'aquesta via *stdin*
> NOTA: La comanda *sed* indicada serveix per quedar-se només amb les línies presents entre les dues indicades (és a dir, el contingut dels certificats de servidor rebuts). Aquest contingut és el que es passarà a la comanda *openssl x509* final (el qual sempre espera, justament, com entrada, un certificat).
> NOTA: Si es volgués veure per pantalla el contingut del certificat "netejat" per *sed* a més de passar-lo a la comanda o*penssl x509* final, un truc seria incloure entubada entre aquestes dues comandes la comanda extra *tee /dev/stderr*

d) Llegeix l'article https://prefetch.net/articles/checkcertificate.html i tot seguit digues per a què serviria la següent comanda: *ssl-cert-check -i -f unfitxer.txt* (la qual pots descarregar de https://github.com/Matty9191/ssl-cert-check) suposant que el contingut de "unfitxer.txt" és aquest:

```
www.google.com 443
www.microsoft.com 443
www.apple.com 443
```

Fins ara només hem estudiat el cas (més habitual) en el què és el servidor qui ha de presentar un certificat als clients que s'hi volen connectar per garantir la seva autenticitat. Però també podem tenir el cas contrari: que siguin els clients qui hagin de presentar al servidor un certificat propi per a què sigui el servidor qui confïi en ells per donar-los accés als recursos que ofereix. Un cas típic d'aquesta altra situació és quan utilitzem l'autenticació per claus en clients SSH (tot i que s'empra el protocol SSH, que és diferent de TLS). En general, si necessitem validar els clients, ens trobarem generalment en el cas de tenir, com abans, un certificat únic del servidor que tots els clients hauran de rebre i verificar (mitjançant algun dels certificats arrel que posseeixin) i, a més, un certificat diferent per cada client que s'hi vulgui connectar a aquest servidor, servidor que els haurà de rebre i verificar una a un segons es vagin connectant mitjançant algun dels certificats arrel que posseeixi).

10.-a) Posa en marxa una màquina virtual (amb la tarja en mode "adaptador pont"), executa allà les següents comandes i digues per a què serveix cadascuna:

```
openssl req -new -x509 -newkey rsa:2048 -nodes -keyout ca.key -out ca.crt      (de CN pots posar qualsevol cosa)
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr    (de CN has de posar la IP de la màq. virt.)
openssl req -new -newkey rsa:2048 -nodes -keyout client1.key -out client1.csr (de CN has de posar la IP de la màq. real)
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -out server.crt -set_serial 1234
openssl x509 -req -in client1.csr -CA ca.crt -CAkey ca.key -out client1.crt -set_serial 5678
```

aII) Copia els fitxers "ca.crt","client1.crt" i "client1.key" a la màquina real (de la manera que vulguis: via *ncat*, *scp*, mail...). Els fitxer "*.csr" els pots esborrar tots si vols.

b) A la màquina virtual executa la comanda següent:
openssl s_server -tls1_3 -key server.key -cert server.crt -port 5555 ***-Verify 10 -CAfile ca.crt***

> NOTA: El paràmetre *-CAfile* és necessari per confiar en el certificat que es rebran dels clients (en no estar la CA signant reconeguda com a tal en el sistema servidor)
> NOTA: El paràmetre *-Verify nº* (amb "V" majúscula!) és imprescindible per obligar al servidor a què demani sempre un certificat al client: si el client en qüestió no el presenta, no s'hi podrà connectar. Sense indicar aquest paràmetre, clients no autenticats es podrien connectar igualment. El número que acompanya aquest paràmetre és la profunditat de la cadena de certificats que es verificarà; no cal pensar-hi gaire: indicant un número relativament elevat com vuit o deu ja està bé.

c) A la màquina real executa la comanda següent (on "x.x.x.x" representa la direcció IP de la màquina virtual):
openssl s_client -connect x.x.x.x:5555 ***-key client1.key -cert client1.crt -CAfile ca.crt***
Comprova que, efectivament, la connexió es realitza sense problemes

> NOTA: El paràmetre *-CAfile* és necessari en aquest cas per confiar en el certificat que es rebrà del servidor (en no estar la CA signant reconeguda com a tal en el sistema client)

cII) Desconnecta el client pulsant CTRL+C ¿Què passa ara si intentes connectar al servidor utilitzant simplement la comanda *openssl s_client -connect x.x.x.x:5555* ?

11.- a) Consulta la documentació del programa gràfic (però internament basat en OpenSSL) anomenat "Xca" (https://github.com/chris2511/xca ; https://www.hohnstaedt.de/xca/index.php) i digues si totes les accions següents les pot realitzar: creació de claus privades; creació de CSRs; creació de certificats autosignats i signatures de CSRs d'altris.

b) A partir de la documentació present a https://github.com/cloudflare/cfssl i a l'article de presentació https://blog.cloudflare.com/introducing-cfssl , digues tres coses que pot fer la comanda *cfssl*

> NOTA: A https://www.pimwiddershoven.nl/entry/install-cfssl-and-cfssljson-cloudflare-kpi-toolkit pots llegir un breu tutorial d'aquesta comanda i a https://prefetch.net/blog/2019/12/10/converting-x509-certificates-to-json-objects pots veure un exemple d'utilitat pràctica

There's three types of certificates:

*__DV-certificates__ (Domain Validation). This is the most basic level of SSL validation. The Certification Authority (CA) only ensure that you are the owner of a specific domain using the information contained in the WHOIS. Typically, the CA exchanges confirmation email with an address listed in the domain's WHOIS record. Alternatively, the CA provides a verification file which the owner places on the website to be protected. Either method confirms that the domain is controlled by the party requesting the certificate. Naturally, this type of certificate enables secure data encryption on your site, but it does not verify that you are the owner of a legitimate business, only of a domain name.

*__OV-certificates__ (Organisation Validation). This kind of certificate authenticates the owner of the site and requires legitimate business information for that company. Its validation process is longer and more detailed: the CA not only verifies the fact that you own the domain, but also the fact that you are the owner of the company. The company must be in a business registry database and in a trusted online directory (for example, dnb.com). Fraudsters cannot get an OV certificate because their organisation cannot be validated. The main advantage of getting an OV-certificate is that legitimate company's data will be listed on the own certificate (in Firefox it can be seen in the "Subject" tab of the certificate window), giving added trust that both the website and the company are reputable. OVs are usually used by corporations, governments and other entities that want to provide an extra layer of confidence to their visitors.

*__EV-certificates__ (Extended Validation). They were supposed to be more trustworthy than OV-certificates but currently they are not used in practice and it seems they won't be.

Note that Let's Encrypt, which is a very popular CA because they issue certificates to everyone who has owns a domain with no charge, only offers DV-certificates. This is primarily because they cannot automate issuance for those types of certificates (they require human effort, which will require paying someone).