325.1 X.509 Certificates and Public Key Infrastructures

Weight: 5

Description: Candidates should understand X.509 certificates and public key infrastructures. They should know how to configure and use OpenSSL to implement certification authorities and issue SSL certificates for various purposes.

Key Knowledge Areas:

- Understand X.509 certificates, X.509 certificate lifecycle, X.509 certificate fields and X.509v3 certificate extensions
- Understand trust chains and public key infrastructures
- Generate and manage public and private keys
- Create, operate and secure a certification authority
- Request, sign and manage server and client certificates
- Revoke certificates and certification authorities

The following is a partial list of the used files, terms and utilities:

- openssl, including relevant subcommands
- OpenSSL configuration
- PEM, DER, PKCS
- CSR
- CRL
- OCSP

En criptografía, **X.509** es un estándar **UIT-T** para infraestructuras de claves públicas (en inglés, ***Public Key Infrastructure*** o **PKI**). **X.509** especifica, entre otras cosas, formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación. Su sintaxis, se define empleando el lenguaje **ASN.1** (**Abstract Syntax Notation One**), y los formatos de codificación más comunes son **DER** (Distinguish Encoding Rules) o **PEM** (**Privacy Enhanced Mail**).

El sistema **X.500** nunca se implementó completamente, y el grupo de trabajo de la infraestructura de clave pública de la IETF, comúnmente conocido como PKIX para *infraestructura de clave pública (**X.509**)* o **PKIX**, adaptó el estándar a la estructura más flexible de Internet. X.509 incluye también estándares para implementación de listas de certificados en revocación (**CRLs**), y con frecuencia aspectos de sistemas **PKI**. De hecho, el término *certificado X.509* se refiere comúnmente al Certificado PKIX y Perfil CRL del certificado estándar de X.509 v3 de la IETF, como se especifica en la RFC 3280.

La forma aprobada por la IETF de chequear la validez de un certificado es el **Online Certificate Status Protocol** (**OCSP**).

Una **autoridad de certificación** (**AC**) es una entidad que emite certificados digitales para su uso por terceros. Es un ejemplo de un tercero de confianza. Las ACs son características en muchos esquemas de infraestructuras de clave pública (**PKI**).

Los certificados raíz de confianza de una organización pueden distribuirse a todos los empleados de manera que ellos puedan usar el sistema de infraestructura de clave pública de la compañía. Navegadores web como Internet Explorer, Netscape/Mozilla y Opera vienen con certificados raíz pre-instalados, de manera que certificados SSL de grandes vendedores que hayan pagado por el privilegio de ser pre-instalados funcionen al instante. En efecto, el propietario del navegador web determina qué ACs son terceros de confianza para los usuarios del navegador web. A pesar de que estos certificados raíz pueden borrarse o deshabilitarse, es muy raro que los usuarios lo hagan.

Existen muchas ACs comerciales que cobran por sus servicios. Instituciones y gobiernos pueden tener sus propias ACs y también existen ACs gratuitas.

En el sistema X.509, una autoridad certificadora (**AC**) emite un certificado asociando una clave pública a un *Nombre Distinguido* particular en la tradición de X.500 o a un *Nombre Alternativo* tal como una dirección de correo electrónico o una entrada de DNS.

Estructura de un certificado

La estructura de un certificado digital X.509 v3 es la siguiente:

- **Certificado**
  - Versión
  - Número de serie del certificado

    - ID del algoritmo utilizado por el CA para firmar (típicamente RSA o DSA)
    - Emisor (CA)
    - Validez
        - No antes de
        - No después de
    - Sujeto, (sujeto titular) expresado en notación DN (Distinguished Name), compuesto por CN (Common Name), OU (Organizational Unit), O (Organization) y C (Country). El sujeto puede ser una persona, un servidor o un servicio.
    - Información de clave pública del sujeto
        - Algoritmo de clave pública
        - Clave pública del sujeto
    - Identificador único de emisor (opcional)
    - Identificador único de sujeto (opcional)
    - Extensiones (opcional)
        - ...
- **Algoritmo usado para firmar el certificado**
- **Firma digital del certificado**

Los identificadores únicos de emisor y sujeto fueron introducidos en la versión 2, y las extensiones en la versión 3. Obsérvese que el número de serie debe ser único para cada certificado emitido por una misma autoridad certificadora (tal como lo indica el RFC 2459).

X.509 es la pieza central de la infraestructura de clave pública y es la estructura de datos que enlaza la clave pública con los datos que permiten identificar al titular. Su sintaxis se define empleando el lenguaje ASN.1 (*Abstract Syntax Notation One*) y los formatos de codificación más comunes son **DER** (***Distinguished Encoding Rules***) o **PEM** (***Privacy-enhanced Electronic Mail***). Siguiendo la notación de ASN.1, un certificado contiene diversos campos, agrupados en tres grandes grupos:

- El primer campo es el *subject* (sujeto), que contiene los datos que identifican al sujeto titular. Estos datos están expresados en notación DN (Distinguished Name), donde un DN se compone a su vez de diversos campos, siendo los más frecuentes los siguientes; CN (*Common Name*), OU (*Organizational Unit*), O (*Organization*) y C (*Country*). Un ejemplo para identificar un usuario mediante el DN, es el siguiente: CN=david.comin O=Safelayer, OU=development, C=ES. Además del nombre del sujeto titular (*subject*), el certificado, también contiene datos asociados al propio certificado digital, como la versión del certificado, su identificador (*serialNumber*), la CA firmante (issuer), el tiempo de validez (*validity*), etc. La versión X.509.v3 también permite utilizar campos opcionales (nombres alternativos, usos permitidos para la clave, ubicación de la CRL y de la CA,etc.).

- En segundo lugar, el certificado contiene la clave pública, que expresada en notación ASN.1, consta de dos campos, en primer lugar, el que muestra el algoritmo utilizado para crear la clave (ej. RSA), y en segundo lugar, la propia clave pública.

- Por último, la CA, ha añadido la secuencia de campos que identifican la firma de los campos previos. Esta secuencia contiene tres atributos, el algoritmo de firma utilizado, el hash de la firma, y la propia firma digital.

Extensiones de archivo de certificados

Las extensiones de archivo de certificados X.509 son:

- `.CER` - Certificado codificado en CER (**Canonical Encoding Rules**), algunas veces es una secuencia de certificados
- `.DER` - Certificado codificado en DER
- `.PEM` - Certificado codificado en **Base64**, encerrado entre **"-----BEGIN CERTIFICATE-----" y "-----END CERTIFICATE-----"**
- `.P7B` - Ver **`.P7c`**
- `.P7C` - Estructura PKCS#7 SignedData sin datos, solo certificado(s) o CRL(s)
- `.PFX` - Ver **`.P12`**
- `.P12` - **PKCS#12**, puede contener certificado(s) (público) y claves privadas (protegido con clave)

PKCS #7 es un estándar para firmar o cifrar datos (ellos lo llaman "sobreado"). Dado que el certificado es necesario para verificar datos firmados, es posible incluirlos en la estructura SignedData. Un archivo `.P7C` es simplemente una estructura SignedData, sin datos para firmar.

PKCS #12 evolucionó del estándar PFX (Personal inFormation eXchange) y se usa para intercambiar objetos públicos y privados dentro de un archivo.

Un archivo **`.PEM`** puede contener certificados o claves privadas, encerrados entre las líneas **BEGIN/END** apropiadas.

Tipos de certificados

Hay distintos tipos de certificados, según el criterio que utilicemos de **Verificación** y **Finalidad**:

Verificación de datos

- **Certificados de Clase 1**: corresponde a los certificados más fáciles de obtener e involucran pocas verificaciones de los datos que figuran en él: sólo el nombre y la dirección de correo electrónico del titular.
- **Certificados de Clase 2**: en los que la Autoridad Certificadora comprueba además el Documento de identidad o permiso de conducir que incluya fotografía, el número de la Seguridad Social y la fecha de nacimiento.

- **Certificados de Clase 3**: en la que se añaden a las comprobaciones de la Clase 2 la verificación de crédito de la persona o empresa mediante un servicio del tipo Equifax, Datacredito.
- **Certificados de Clase 4**: que a todas las comprobaciones anteriores suma la verificación del cargo o la posición de una persona dentro de una organización.

Finalidad

- **Certificados SSL para cliente**: mediante el protocolo Secure Socket Layer, dirigido a una persona física.
- **Certificados SSL para servidor**: usados para identificar a un servidor ante un cliente en comunicaciones mediante SSL.
- **Certificados S/MIME**: usados para servicios de correo electrónico firmado y cifrado, que se expiden generalmente a una persona física.
- **Certificados para la firma de código**: usados para identificar al autor de ficheros o porciones de código en cualquier lenguaje de programación que se deba ejecutar en red (Java, JavaScript, CGI, etc).
- **Certificados para AC (Autoridades Certificadoras)**: se usa por el software cliente para determinar si pueden confiar en un certificado cualquiera, accediendo al certificado de la AC y comprobando que ésta es de confianza.

Ejemplo de certificado X.509 y del proceso de validación

Como ejemplo de un certificado X.509, se encuentra aquí una decodificación (generada con openssl) de uno de los certificados viejos de `www.freesoft.org`. El certificado real tiene un tamaño de alrededor de 1KB. Fue emitido (firmado) por Thawte (desde que fue adquirido por Verisign), tal como se indica en el campo Emisor. El tema contiene bastante información personal, pero la parte más importante es el nombre común (CN) de `www.freesoft.org` - esta es la parte que debe coincidir con la terminal que se está autenticando. A continuación viene una clave pública RSA (módulo y exponente público), seguido de la firma, computada tomando un hash MD5 de la primera parte del certificado y cifrándola con la clave privada RSA de Thawte.

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
```

```
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

Obsérvese que al final del certificado se encuentra la firma de mismo. Para estampar esta firma, la autoridad certificadora calcula un hash MD5 de la primera parte del certificado (la sección de *Data*: los datos del mismo más la clave pública) y cifra ese hash con su clave privada, que mantiene en secreto. Ahora supongamos que un cliente se conecta a *www.freesoft.org* y el sitio le devuelve el certificado anterior, con la intención de probar su identidad. Para validar este certificado, necesitamos el certificado de la autoridad certificadora, que fue el emisor del primero (Thawte Server CA). Se toma la clave pública del certificado de la autoridad certificadora para decodificar la firma del primer certificado, obteniéndose un hash MD5. Este hash MD5 debe coincidir con el hash MD5 calculado sobre la primera parte del certificado. En ese caso el proceso de validación termina con éxito. Si no, la validación no tiene éxito, y no puede asegurarse que el certificado de *www.freesoft.org* está vinculado con esa clave pública. Dicho de otro modo, es posible que *www.freesoft.org* no sea quien dice ser. A continuación se muestra el certificado de la CA:

```
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 1 (0x1)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Aug  1 00:00:00 1996 GMT
           Not After : Dec 31 23:59:59 2020 GMT
       Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
       Subject Public Key Info:
```

```
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
                    68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
                    85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
                    6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
                    6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
                    29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
                    6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
                    5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
                    3a:c2:b5:66:22:12:d6:87:0d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
        a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
        3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
        4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
        8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
        e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
        b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
        70:47
```

Este es un ejemplo de un certificado auto firmado. Nótese que el CN del Emisor y el CN del Sujeto son iguales. No hay forma de verificar este certificado, salvo que se comprueba contra sí mismo. En este caso, hemos alcanzado el fin de la cadena de certificados. ¿Cómo es entonces que este certificado se hace confiable? Simple: se configura manualmente. Thawte es una de las autoridades certificantes raíz reconocida por Microsoft y Netscape. Este certificado ya viene con el navegador web (probablemente pueda encontrarse listado como "Thawte Server CA" en las opciones de seguridad) y es confiable por defecto. Como certificado confiado globalmente de larga vida (nótese la fecha de vencimiento) que puede firmar prácticamente cualquier cosa (nótese la falta de limitaciones), su clave privada correspondiente debe ser una de las más ocultas del mundo.

OpenSSL

OpenSSL es un proyecto de código abierto que consiste en una biblioteca criptográfica y un SSL / TLS. OpenSSL tiene licencia dual bajo licencias OpenSSL y SSLeay.

↘ Determinar la Versión Instalada

[root@selinux ~]# openssl version
OpenSSL 1.0.1e-fips 11 Feb 2013

[root@selinux ~]# openssl version -a

OpenSSL 1.0.1e-fips 11 Feb 2013

built on: Mon Feb 20 14:38:48 UTC 2017

platform: linux-x86_64

options: bn(64,64) md2(int) rc4(16x,int) des(idx,cisc,16,int) idea(int) blowfish(idx)

compiler: gcc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DKRB5_MIT -m64 -DL_ENDIAN -DTERMIO -Wall -O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -Wa,--noexecstack -DPURIFY -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM

OPENSSLDIR: "/etc/pki/tls"

engines: rdrand dynamic

La última línea remarcada es muy interesante porque nos indica la ruta de configuración y ubicación de los certificados → ‘**/etc/pki/tls’.**

[root@selinux ~]# ls -al /etc/pki/tls/

drwxr-xr-x. 5 root root 76 may 10 12:06 .

drwxr-xr-x. 9 root root 91 nov 5 2016 ..

lrwxrwxrwx. 1 root root 49 may 10 12:06 cert.pem -> /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem

drwxr-xr-x. 2 root root 112 may 10 12:06 certs

drwxr-xr-x. 2 root root 69 mar 16 07:50 misc

-rw-r--r--. 1 root root 10923 feb 20 15:30 openssl.cnf

drwxr-xr-x. 2 root root 6 feb 20 15:40 private

La carpeta **misc/** contiene algunos scripts suplementarios, los más interesantes son los scripts que permiten implementar una autoridad de certificación privada (CA).

[root@selinux ~]# ls -al /etc/pki/tls/misc/

drwxr-xr-x. 2 root root 69 mar 16 07:50 .

drwxr-xr-x. 5 root root 76 may 10 12:06 ..

```
-rwxr-xr-x. 1 root root 5178 feb 20 15:40 CA
-rwxr-xr-x. 1 root root  119 feb 20 15:40 c_hash
-rwxr-xr-x. 1 root root  152 feb 20 15:40 c_info
-rwxr-xr-x. 1 root root  112 feb 20 15:40 c_issuer
-rwxr-xr-x. 1 root root  110 feb 20 15:40 c_name
```

↘ *Compilar OpenSSL*

*(Note: The latest stable version is the **1.1.0** series of releases. Also available is the 1.0.2 series. This is also our Long Term Support (**LTS**) version (support will be provided until 31st December 2019). The 0.9.8, 1.0.0 and 1.0.1 versions are now out of support and should not be used).*

```
[root@selinux tmp]# wget -c https://www.openssl.org/source/openssl-1.1.0e.tar.gz
[root@selinux tmp]# wget -c https://www.openssl.org/source/openssl-1.1.0e.tar.gz.sha256
[root@selinux tmp]# cat openssl-1.1.0e.tar.gz.sha256
57be8618979d80c910728cfc99369bf97b2a1abd8f366ab6ebdee8975ad3874c
[root@selinux tmp]# sha256sum openssl-1.1.0e.tar.gz
57be8618979d80c910728cfc99369bf97b2a1abd8f366ab6ebdee8975ad3874c openssl-1.1.0e.tar.gz
[root@selinux tmp]# tar xvzf openssl-1.1.0e.tar.gz
[root@selinux openssl-1.1.0e]# ./config \
> --prefix=/opt/openssl \
> --openssldir=/opt/openssl \
> enable-ec_nistp_64_gcc_128
[root@selinux openssl-1.1.0e]# yum install gcc
[root@selinux openssl-1.1.0e]# make depend && make && make install
[root@selinux openssl]# ls /opt/openssl/
bin  certs  include  lib  misc  openssl.cnf  openssl.cnf.dist  private  share
```

↘ *Comandos disponibles*

```
[root@selinux openssl]# openssl help
openssl:Error: 'help' is an invalid command.
```

Standard commands

asn1parse ca ciphers cms

crl crl2pkcs7 dgst dh

dhparam dsa dsaparam ec

ecparam enc engine errstr

gendh gendsa genpkey genrsa

nseq ocsp passwd pkcs12

pkcs7 pkcs8 pkey pkeyparam

pkeyutl prime rand req

rsa rsautl s_client s_server

s_time sess_id smime speed

spkac ts verify version

x509

Message Digest commands (see the `**dgst**' command for more details)

md2 md4 md5 rmd160

sha sha1

Cipher commands (see the `**enc**' command for more details)

aes-128-cbc aes-128-ecb aes-192-cbc aes-192-ecb

aes-256-cbc aes-256-ecb base64 bf

bf-cbc bf-cfb bf-ecb bf-ofb

camellia-128-cbc camellia-128-ecb camellia-192-cbc camellia-192-ecb

camellia-256-cbc camellia-256-ecb cast cast-cbc

cast5-cbc cast5-cfb cast5-ecb cast5-ofb

des des-cbc des-cfb des-ecb

des-ede des-ede-cbc des-ede-cfb des-ede-ofb

des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb

des-ofb des3 desx idea

idea-cbc idea-cfb idea-ecb idea-ofb

rc2 rc2-40-cbc rc2-64-cbc rc2-cbc

rc2-cfb rc2-ecb rc2-ofb rc4

rc4-40 seed seed-cbc seed-cfb

seed-ecb seed-ofb zlib

[root@selinux openssl]# openssl dgst --help

unknown option '--help'

options are

-c to output the digest with separating colons

-r to output the digest in coreutils format

-d to output debug info

-hex output as hex dump

-binary output in binary form

-sign file sign digest using private key in file

-verify file verify a signature using public key in file

-prverify file verify a signature using private key in file

-keyform arg key file format (PEM or ENGINE)

-out filename output to filename rather than stdout

-signature file signature to verify

-sigopt nm:v signature parameter

-hmac key create hashed MAC with key

-mac algorithm create MAC (not neccessarily HMAC)

-macopt nm:v MAC algorithm parameters or key

-engine e use engine e, possibly a hardware device.

-md4 to use the md4 message digest algorithm

-md5 to use the md5 message digest algorithm

-ripemd160 to use the ripemd160 message digest algorithm

-sha to use the sha message digest algorithm

-sha1 to use the sha1 message digest algorithm

-sha224 to use the sha224 message digest algorithm

-sha256 to use the sha256 message digest algorithm

-sha384 to use the sha384 message digest algorithm

-sha512 to use the sha512 message digest algorithm

-whirlpool to use the whirlpool message digest algorithm

[root@selinux openssl]# openssl enc --help

unknown option '--help'

options are

-in <file> input file

-out <file> output file

-pass <arg> pass phrase source

-e encrypt

-d decrypt

-a/-base64 base64 encode/decode, depending on encryption flag

-k passphrase is the next argument

-kfile passphrase is the first line of the file argument

-md the next argument is the md to use to create a key

from a passphrase. See openssl dgst -h for list.

-S salt in hex is the next argument

-K/-iv key/iv in hex is the next argument

-[pP] print the iv/key (then exit if -P)

-bufsize <n> buffer size

-nopad disable standard block padding

-engine e use engine e, possibly a hardware device.

Cipher Types

-aes-128-cbc -aes-128-cbc-hmac-sha1 -aes-128-cfb

-aes-128-cfb1 -aes-128-cfb8 -aes-128-ctr

-aes-128-ecb -aes-128-gcm -aes-128-ofb

-aes-128-xts -aes-192-cbc -aes-192-cfb

-aes-192-cfb1 -aes-192-cfb8 -aes-192-ctr

-aes-192-ecb -aes-192-gcm -aes-192-ofb

-aes-256-cbc -aes-256-cbc-hmac-sha1 -aes-256-cfb

-aes-256-cfb1 -aes-256-cfb8 -aes-256-ctr

-aes-256-ecb -aes-256-gcm -aes-256-ofb

-aes-256-xts -aes128 -aes192

-aes256 -bf -bf-cbc

-bf-cfb -bf-ecb -bf-ofb

-blowfish -camellia-128-cbc -camellia-128-cfb

-camellia-128-cfb1 -camellia-128-cfb8 -camellia-128-ecb

-camellia-128-ofb -camellia-192-cbc -camellia-192-cfb

-camellia-192-cfb1 -camellia-192-cfb8 -camellia-192-ecb

-camellia-192-ofb -camellia-256-cbc -camellia-256-cfb

-camellia-256-cfb1 -camellia-256-cfb8 -camellia-256-ecb

-camellia-256-ofb -camellia128 -camellia192

-camellia256 -cast -cast-cbc

-cast5-cbc -cast5-cfb -cast5-ecb

-cast5-ofb -des -des-cbc

-des-cfb -des-cfb1 -des-cfb8

-des-ecb -des-ede -des-ede-cbc

-des-ede-cfb -des-ede-ofb -des-ede3

-des-ede3-cbc -des-ede3-cfb -des-ede3-cfb1

-des-ede3-cfb8 -des-ede3-ofb -des-ofb

-des3 -desx -desx-cbc

-id-aes128-GCM -id-aes128-wrap -id-aes128-wrap-pad

-id-aes192-GCM -id-aes192-wrap -id-aes192-wrap-pad

-id-aes256-GCM -id-aes256-wrap -id-aes256-wrap-pad

-id-smime-alg-CMS3DESwrap -idea -idea-cbc

-idea-cfb -idea-ecb -idea-ofb

-rc2 -rc2-40-cbc -rc2-64-cbc

-rc2-cbc -rc2-cfb -rc2-ecb

-rc2-ofb -rc4 -rc4-40

-rc4-hmac-md5 -seed -seed-cbc

-seed-cfb -seed-ecb -seed-ofb

↘ ***Conversión de Mozilla en -Trust Store- utilizando PERL (Proyecto CURL)***

[root@selinux tmp]# wget -c https://raw.githubusercontent.com/bagder/curl/master/lib/mk-ca-bundle.pl

[root@selinux tmp]# chmod +x mk-ca-bundle.pl

[root@selinux tmp]# ./mk-ca-bundle.pl

SHA256 of old file: 0

Downloading certdata.txt ...

Get certdata with curl!

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1581k  100 1581k    0     0   119k      0  0:00:13  0:00:13 --:--:--  319k
```

Downloaded certdata.txt

SHA256 of new file: dffa79e6aa993f558e82884abf7bb54bf440ab66ee91d82a27a627f6f2a4ace4

Processing 'certdata.txt' ...

Done (158 CA certs processed, 34 skipped).

[root@selinux tmp]# ls

ca-bundle.crt certdata.txt mk-ca-bundle.pl

[root@selinux tmp]# cat certdata.txt

…

Trust for "Symantec Class 1 Public Primary Certification Authority - G4"

Issuer: CN=Symantec Class 1 Public Primary Certification Authority - G4,OU=Symantec Trust Network,O=Symantec Corporation,C=US

Serial Number:21:6e:33:a5:cb:d3:88:a4:6f:29:07:b4:27:3c:c4:d8

Subject: CN=Symantec Class 1 Public Primary Certification Authority - G4,OU=Symantec Trust Network,O=Symantec Corporation,C=US

Not Valid Before: Wed Oct 05 00:00:00 2011

Not Valid After : Mon Jan 18 23:59:59 2038

Fingerprint (SHA-256): 36:3F:3C:84:9E:AB:03:B0:A2:A0:F6:36:D7:B8:6D:04:D3:AC:7F:CF:E2:6A:0A:91:21:AB:97:95:F6:E1:76:DF

Fingerprint (SHA1): 84:F2:E3:DD:83:13:3E:A9:1D:19:52:7F:02:D7:29:BF:C1:5F:E6:67

CKA_CLASS CK_OBJECT_CLASS CKO_NSS_TRUST

CKA_TOKEN CK_BBOOL CK_TRUE

CKA_PRIVATE CK_BBOOL CK_FALSE

CKA_MODIFIABLE CK_BBOOL CK_FALSE

CKA_LABEL UTF8 "Symantec Class 1 Public Primary Certification Authority - G4"

CKA_CERT_SHA1_HASH MULTILINE_OCTAL

\204\362\343\335\203\023\076\251\035\031\122\177\002\327\051\277

\301\137\346\147

END

CKA_CERT_MD5_HASH MULTILINE_OCTAL

\004\345\200\077\125\377\131\207\244\062\322\025\245\345\252\346

…

Administración del ‘Certificado’ y de la ‘Key’

La mayoría de los usuarios recurren a OpenSSL porque desean configurar y ejecutar un servidor web que soporte SSL. Ese proceso consta de tres pasos:

(**1**) Generar una fuerte y segura clave privada.
(**2**) Crear un **Certificado Signing Request** (**CSR**) y enviarlo a una **CA**.
(**3**) Instalar la **CA** suministrada y el **certificado** en su servidor web.

1 ↘ *Generar la Key*

→ Algoritmo Key:

OpenSSL admite claves **RSA**, **DSA** y **ECDSA**, pero no todos los tipos son prácticos para su uso en todos los escenarios.

→ → **RSA** (***Rivest***, ***Shamir*** *y* ***Adleman***) es un sistema criptográfico de clave pública desarrollado en 1977. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente. La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de , y se prevé que su tamaño crezca con el aumento de la capacidad de cálculo de los ordenadores.
Como en todo sistema de clave pública, cada usuario posee dos claves de cifrado: una pública y otra privada. Cuando se quiere enviar un mensaje, el emisor busca la clave pública del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, este se ocupa de descifrarlo usando su clave privada.
Se cree que RSA será seguro mientras no se conozcan formas rápidas de descomponer un número grande en producto de primos. La computación cuántica podría proveer de una solución a este problema de factorización.
→ → **DSA** (**Digital Signature Algorithm)**, en español → **Algoritmo de Firma Digital** es un estándar del Gobierno Federal de los Estados Unidos de América o FIPS para firmas digitales. Fue un Algoritmo propuesto por el Instituto Nacional de Normas y Tecnología de los Estados Unidos para su uso en su Estándar de Firma Digital (DSS), especificado en el FIPS 186. DSA se hizo público el 30 de agosto de 1991, este algoritmo como su nombre lo indica, **sirve para firmar y no para cifrar información**. Una desventaja de este algoritmo es que requiere mucho más tiempo de cómputo que RSA.
→ → **ECDSA**. (**Elliptic Curve Digital Signature Algorithm**) es una modificación del algoritmo DSA que emplea operaciones sobre puntos de curvas elípticas en lugar de las exponenciaciones que usa DSA (problema del logaritmo discreto). La principal ventaja de este esquema es que requiere números de tamaños menores para brindar la misma seguridad que DSA o RSA. Existen dos tipos de curvas dependiendo del campo finito en el que se definan, que pueden ser GF(P) o GF(2^m).

Por ejemplo, para las claves del **servidor web**, todos utilizan → **RSA**, ya que las claves DSA estan limitadas a 1.024 bits (Internet Explorer no soporta nada más fuerte).
Las claves ECDSA aún no han sido ampliamente apoyadas por las CA's. Para **SSH** → **DSA** y **RSA** son ampliamente utilizadas, mientras que ECDSA no esta soportado para todos los clientes.

→ Tamaño de la Key:
Hoy en día, las claves **RSA** de 2,048 bits se consideran seguras, y eso es lo que debe se debería utilizar. Recomendable utilizar 2,048 bits para las claves **DSA** y al menos 256 bits para **ECDSA**.

→ Passphrase:
El uso de una frase de contraseña con una clave es opcional, pero muy recomendable. Se pueden transportar, almacenar, … Por otro lado, esas claves pueden ser un inconveniente, porque no se

pueden usar sin sus frase de acceso. Por ejemplo, se pedirá teclear la frase de acceso cada vez que se reinicie el servidor web. Además, el uso de claves protegidas en producción no aumenta realmente la seguridad. Esto se debe a que, una vez activadas, las claves privadas se mantienen desprotegidas en memoria. Un atacante que pueda llegar al servidor puede obtenerlas desde alli. Por lo tanto, las frases clave deben ser vistas sólo como un mecanismo para proteger claves privadas cuando no están instaladas en sistemas de producción. Para una mayor seguridad en producción debe plantearse la necesidad de invertir en una solución de hardware.

↘ ↘***Generar Key 'RSA' → (Rivest, Shamir y Adleman)***

[root@selinux tls]# ll

lrwxrwxrwx. 1 root root 49 may 10 12:06 cert.pem -> /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem

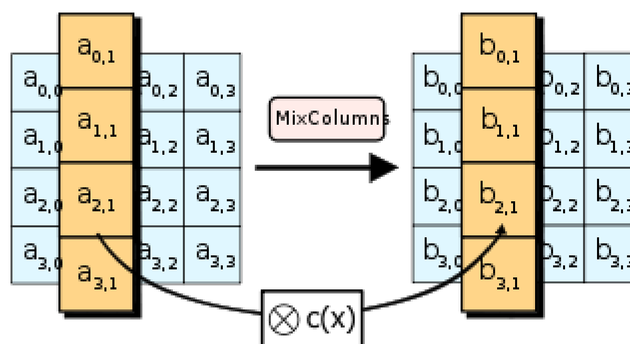
drwxr-xr-x. 2 root root 112 may 10 12:06 certs

drwxr-xr-x. 2 root root 69 mar 16 07:50 misc

-rw-r--r--. 1 root root 10923 feb 20 15:30 openssl.cnf

drwxr-xr-x. 2 root root 6 feb 20 15:40 private

→ **Advanced Encryption Standard** (**AES**), también conocido como **Rijndael** (pronunciado "Rain Doll" en inglés), es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. El AES fue anunciado por el Instituto Nacional de Estándares y Tecnología (NIST) como FIPS PUB 197 de los Estados Unidos (FIPS 197) el 26 de noviembre de 2001 después de un proceso de estandarización que duró 5 años. Se transformó en un estándar efectivo el 26 de mayo de 2002. Desde 2006, el AES es uno de los algoritmos más populares usados en criptografía simétrica.



→ **Generamos la Private Key**

[root@selinux tls]# openssl genrsa -aes128 -out fd.key 2048

Generating **RSA private key, 2048** bit long modulus

.............+++

....+++

e is 65537 (0x10001)

Enter pass phrase for fd.key:

Verifying - Enter pass phrase for fd.key:

[root@selinux tls]# ll

lrwxrwxrwx. 1 root root 49 may 10 12:06 cert.pem -> /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem

drwxr-xr-x. 2 root root 112 may 10 12:06 certs

-rw-r--r--. 1 root root 1766 may 11 01:32 fd.key → Key Privada

drwxr-xr-x. 2 root root 69 mar 16 07:50 misc

-rw-r--r--. 1 root root 10923 feb 20 15:30 openssl.cnf

drwxr-xr-x. 2 root root 6 feb 20 15:40 private

[root@selinux tls]# cat fd.key

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,C8089A3F57E8F253B86E9C906D6B8F43

aRGDgi2U0L6Ug2lh3llWG1plia/7OOfI61awfEpC+rA2+rl4OVW0rOvUSzEZOW0g

KtRNGi9zsfwB+kBNbkn9VGHYNfB69E0LIXw5KxHbwdZT0fS+Lq5Ni5ZpWZUgu4nr

...

-----END RSA PRIVATE KEY-----

→ Mostramos la estructura de la Key

[root@selinux tls]# openssl rsa -text -in fd.key

Enter pass phrase for fd.key:

Private-Key: (2048 bit)

modulus:

00:91:a6:de:3f:be:cb:3a:e7:84:1d:7c:c8:9a:37:

0f:c0:7c:0b:d1:ff:07:56:35:fe:03:20:ee:2d:d6:

...

prime1:

00:c1:98:bc:48:91:52:8d:e7:3f:ef:68:ac:48:7e:

d1:e4:f0:7e:46:8b:36:be:9e:a3:e3:e9:46:c8:f1:

...

prime2:

00:c0:99:cf:47:ca:16:8a:98:a5:98:96:85:8f:95:

2e:03:f3:08:5a:f3:73:5c:9a:7b:0d:fa:27:16:13:

...

exponent1:

00:99:d5:2f:e7:c9:f1:fb:68:41:d9:8f:27:37:03:

ee:ed:28:5c:6e:d9:b8:4a:87:ec:5a:f1:c6:99:6d:

...

exponent2:

00:bc:83:4f:a1:02:aa:41:89:db:3b:98:c8:ad:9a:

e4:69:35:2d:8e:68:0f:18:2a:94:1b:40:27:95:b2:

...

coefficient:

2c:fe:02:be:fb:85:5d:77:7d:0f:e7:46:3b:82:3a:

9c:fe:41:eb:ef:06:63:b8:ce:a8:95:0c:6c:21:02:

...

writing RSA key

-----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEAkabeP77LOueEHXzImjcPwHwL0f8HVjX+AyDuLdZc79WdnDbe

...

-----END RSA PRIVATE KEY-----

→ **Generamos la Public Key**

[root@selinux tls]# openssl rsa -in fd.key -pubout -out fd-public.key

Enter pass phrase for fd.key:

writing RSA key

[root@selinux tls]# ll

lrwxrwxrwx. 1 root root 49 may 10 12:06 cert.pem -> /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem

drwxr-xr-x. 2 root root 112 may 10 12:06 certs

-rw-r--r--. 1 root root 1766 may 11 01:32 fd.key

-rw-r--r--. 1 root root 1679 may 11 01:56 **fd-public.key** → **Key Pública**

drwxr-xr-x. 2 root root 69 mar 16 07:50 misc

-rw-r--r--. 1 root root 10923 feb 20 15:30 openssl.cnf

drwxr-xr-x. 2 root root 6 feb 20 15:40 private

[root@selinux tls]# cat fd-public.key

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkabeP77LOueEHXzImjcP

...

TP2SQIZCOC0m+I94OCXuQ0m5JkJJotO+VVBtdyTl07dGIXg936FE0TMhUVeNxvd6

fQIDAQAB

-----END PUBLIC KEY-----

↘ ↘*Generar Key 'DSA' → (Digital Signature Algorithm)*

La generación de la Key DSA se realiza en 2 pasos con el comando 'openssl': Primero → **creamos los parámetros DSA**, y en Segundo lugar → **generamos la Key**.

→ **Generamos la Private Key**

[root@selinux tls]# openssl dsaparam -genkey 2048 | openssl dsa -out dsa.key -aes128

read DSA key

Generating DSA parameters, 2048 bit long prime

This could take some time

```
........+.....+.......+..+..+..+.....................+..+...+......+....+................+............+.......+........
+.........................................+.....................+....+.+.................+.+.........+...........................+...
+...............+.+.+......+........................+.........+...............+..........................+......+......
+.........................+..........................+.....+...........+.....+.................+..........+.........+........+....
+.........................+...........+..+........+..............................+....+......+............+.............+....
+.................................+..+..+........+........+..+.............+..........+........+.+..+........+........+..........+..
+..+.................+...........+.........................+.+.....+.........................+......++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++*

........+............+........+..........+...............+......+...........+................+....+.......+..............+...+..........
+.......+........+....+...............................+.+........+.....+.+...+.............+.................+........
+.........................+.+......+...................+........+.........+...+.......+.....+....+..+..+................
+........................+.....+.....+..........+...........+.+...+..........+....+........+.+.........................
+............................+...............+....................+.....+....+.........+......+....+++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++*
```

writing DSA key

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

[root@selinux tls]# ls

cert.pem certs **dsa.key** fd.key fd-public.key misc openssl.cnf private

[root@selinux tls]# cat dsa.key

-----BEGIN DSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,862EF827074E33804F694CEDB3A03BBB

pf85ASQdOHPy2i4FmadXuArpw6Oviu7niiQvVpML/VzaVRu03tNBfH8uTrKRvAAl

...

Z3HboevxP+Wtk6Bf6zjuj6zmTvg0mOOsCLXWzLavCbZ95RSC1+ZLCCgOBPXzxLWC

-----END DSA PRIVATE KEY-----

→ **Generamos la Public Key**

[root@selinux tls]# openssl dsa -in dsa.key -pubout -out dsa-public.key

read DSA key

Enter pass phrase for dsa.key:

writing DSA key

[root@selinux tls]# ls

cert.pem certs dsa.key **dsa-public.key** fd.key fd-public.key misc openssl.cnf private

[root@selinux tls]# cat dsa-public.key

-----BEGIN PUBLIC KEY-----

MIIDRjCCAjkGByqGSM44BAEwggIsAoIBAQDK05RjrCb6g1vQYZ675Kw+8llHpV3Z

...

UGJDUe5gSLAl8Rr8Q7kX3jG4gOle4gLbguU7IhbUy4egHz9yp7sH8/oNYjAAqxaJ

98cjwu72iJqeC4hLfyHo40mKU1++pdZ7Wg8=

-----END PUBLIC KEY-----

↘ ↘*Generar Key 'ECDSA' → (Elliptic Curve Digital Signature Algorithm)*

Para **ECDSA** no es posible crear Key's de tamaño arbitrario. Cada Key debe seleccionar un nombre de parámetro de curva elíptica (EC), la cual controla el tamaño y otras características EC.

→ **Listar las EC soportadas (Elliptic Curve Operations)**

[root@selinux tls]# openssl ecparam -list_curves

secp384r1 : NIST/SECG curve over a 384 bit prime field

secp521r1 : NIST/SECG curve over a 521 bit prime field

prime256v1: X9.62/SECG curve over a 256 bit prime field

Para Servidores Web las las Key's están limitadas solamente a 2 tipos de curvas: '**secp384r1**' y '**prime256v1**'.

→ **Generamos la Private Key**

[root@selinux tls]# openssl ecparam -genkey -name prime256v1 | openssl ec -out ec.key -aes128

read EC key

writing EC key

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

[root@selinux tls]# cat ec.key

-----BEGIN EC PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,89FDA259496624E8FE7D027E099FD273

rNgWb4hGe3iRXwPR4EHBwgKf8X+nvLz1lBgltXCjMu2VxyFxbOpdSGbOHPMdjrP+

iXrviWq2vWsdJ38ID12y8LcpCCyUF5TSpaSvN7SfaxAAtQZt3bsUI8ZGm6MWP2gJ

RnO611c8b+b1RUh2o0Uhhj5UWXpuDdEjfML3ZEaaP/4=

-----END EC PRIVATE KEY-----

→ **Generamos la Public Key**

[root@selinux tls]# openssl ec -in ec.key -pubout -out ec-public.key

read EC key

Enter PEM pass phrase:

writing EC key

[root@selinux tls]# cat ec-public.key

-----BEGIN PUBLIC KEY-----

MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEgwiMRnJAqc9flqDlWUc+jEuUS2ro

da5q2ESk8PeIiPV1PtYHzw93xdDGg+Rf/DI7BeSvOoS8rFM1CLUsfkGoPg==

-----END PUBLIC KEY-----

2 ↘ *Crear un Certificate Signing Requests → CSR*

↘↘ *Crear un nuevo CSR*

Al disponer de una **Private Key** puede crearse ya un **Certificate Signing Request** (**CSR**). Contiene la **Public Key** firmada con la **Private Key** ademas de otra información de la entidad.

Normalmente es un proceso interactivo en el que se suministra información de la entidad. Si no se desea introducir información debe indicarse con un **punto** '**.**' y **<enter>**. Informa con valores por defecto de acuerdo a su fichero de configuración →'**/etc/pki/tls/openssl.cnf**' , o tambien puede proporcionarse uno personalizado.

[root@selinux tls]# openssl req -new -key fd.key -out fd.csr

Enter pass phrase for fd.key:

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [XX]:ES

State or Province Name (full name) []:**.**

Locality Name (eg, city) [Default City]:Gondomar

Organization Name (eg, company) [Default Company Ltd]:cadilinea, slu

Organizational Unit Name (eg, section) []:**.**

Common Name (eg, your name or your server's hostname) []:www.cadilinea.com

Email Address []:info@cadilinea.com

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:

[root@selinux tls]# cat /etc/pki/tls/fd.csr

-----BEGIN CERTIFICATE REQUEST-----

MIICvTCCAaUCAQAweDELMAkGA1UEBhMCRVMxETAPBgNVBAcMCEdvbmRvbWFyMR

...

wYjLD97rgN4qeDfZsDNkI2D/7zEo96NrlwKkjQiGAfE2

-----END CERTIFICATE REQUEST-----

[root@selinux tls]# openssl req -text -in fd.csr -noout

Certificate Request:

Data:

```
        Version: 0 (0x0)
        Subject: C=ES, L=Gondomar, O=cadilinea, slu, CN=www.cadilinea.com/emailAddress=info@cadilinea.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:91:a6:de:3f:be:cb:3a:e7:84:1d:7c:c8:9a:37:
                  ...
                    46:21:78:3d:df:a1:44:d1:33:21:51:57:8d:c6:f7:
                    7a:7d
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         4f:0c:c9:1f:cf:49:06:a4:68:ff:49:b2:24:d8:30:ca:a8:0a:
       ...
         d9:b0:33:64:23:60:ff:ef:31:28:f7:a3:6b:97:02:a4:8d:08:
         86:01:f1:36
```

↘↘ ***Crear un nuevo CSR en modo -Desatendido-***

La generación de los CSR puede realizarse de forma desatendida a través de un fichero de configuración.

Par generar un CSR para '**www.feistyduck.com**' por ejemplo, necesitamos un fichero de configuración que denominamos → '**fd-01.cnf**' con la siguiente configuración:

```
[req]
prompt = no
distinguished_name = dn
```

req_extensions = ext

input_password = **clave_de_fd-01.key**

[dn]

CN = www.feistyduck.com

emailAddress = webmaster@feistyduck.com

O = Feisty Duck Ltd

L = London

C = GB

[ext]

subjectAltName = DNS:www.feistyduck.com,DNS:feistyduck.com

[root@selinux tls]# openssl req -new -config fd-01.cnf -key fd-01.key -out fd-01.csr

[root@selinux tls]# cat fd-01.csr

-----BEGIN CERTIFICATE REQUEST-----

MIIDAzCCAesCAQAwfjEbMBkGA1UEAxMSd3d3LmZlaXN0eWR1Y2suY29tMScwJQYJ

KoZIhvcNAQkBFhh3ZWJtYXN0ZXJAZmVpc3R5ZHVjay5jb20xGDAWBgNVBAoTD0Zl

...

BYspZe6QWWuD9WAouVJy+RsHBkCaQ7bCRSJ+IXBCRhTRxpRT0OUTtgz8OwnEftLe

c9MzTbozdQ==

-----END CERTIFICATE REQUEST-----

[root@selinux tls]# openssl req -text -in fd-01.csr -noout

Certificate Request:

Data:

Version: 0 (0x0)

Subject: CN=www.feistyduck.com/emailAddress=webmaster@feistyduck.com, O=Feisty Duck Ltd, L=London, C=GB

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

00:91:a6:de:3f:be:cb:3a:e7:84:1d:7c:c8:9a:37:

...

46:21:78:3d:df:a1:44:d1:33:21:51:57:8d:c6:f7:

7a:7d

Exponent: 65537 (0x10001)

Attributes:

Requested Extensions:

X509v3 Subject Alternative Name:

DNS:www.feistyduck.com, DNS:feistyduck.com

Signature Algorithm: sha1WithRSAEncryption

6d:90:44:60:9f:a1:4c:ef:10:bd:5a:d3:a5:ea:f7:3e:55:2c:

...

c6:94:53:d0:e5:13:b6:0c:fc:3b:09:c4:7e:d2:de:73:d3:33:

4d:ba:33:75

3 ↘ *Firmar Certificados → CRT*

↘↘ *Crear el CRT a partir del CSR previamente generado*

[root@selinux tls]# openssl x509 -req -days 365 -in fd.csr -signkey fd.key -out fd.crt

Signature ok

subject=/C=ES/L=Gondomar/O=cadilinea, slu/CN=www.cadilinea.com/emailAddress=info@cadilinea.com

Getting Private key

Enter pass phrase for fd.key:

[root@selinux tls]# cat fd.crt

-----BEGIN CERTIFICATE-----

MIIDbDCCAlQCCQDcccwPtTa8ZjANBgkqhkiG9w0BAQUFADB4MQswCQYDVQQGEwJF

UzERMA8GA1UEBwwIR29uZG9tYXIxFzAVBgNVBAoMDmNhZGlsaW5lYSwgc2x1MRow

GAYDVQQDDBF3d3cuY2FkaWxpbmVhLmNvbTEhMB8GCSqGSIb3DQEJARYSaW5mb0Bj

...

mfEVmrC1AeK78J18HJueEPEQkJFUxzeXt50CiVieG/bPX3xYmwM5qOKj1j1EKx6P

YlVRSffZjWupSy3JElwZcA==

-----END CERTIFICATE-----

↘↘ *Crear el CRT de una sola vez de forma interactiva*

En realidad, no es necesario crear un '**CSR**' y firmar en dos pasos independientes. El siguiente comando crea el '**CRT'** directamente, sin necesidad de generar previamente el '**CSR**':

[root@selinux tls]# openssl req -new -x509 -days 365 -key fd.key -out fd.crt

Enter pass phrase for fd.key:

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [XX]:ES

State or Province Name (full name) []:.

Locality Name (eg, city) [Default City]:Gondomar

Organization Name (eg, company) [Default Company Ltd]:cadilinea, slu

Organizational Unit Name (eg, section) []:.

Common Name (eg, your name or your server's hostname) []:cadilinea.com

Email Address []:info@cadilinea.com

↘↘ Crear el CRT de una sola vez de forma NO interactiva → '-subj'

**$ openssl req -new -x509 -days 365 -key fd.key -out fd.crt **

-subj "/C=GB/L=London/O=Feisty Duck Ltd/CN=www.feistyduck.com"

----→ Empezar PARTE-2 y Depurar ideas.

↘↘ Crear el CRS para un CRT existente

↘↘ Crear CRS's para un CRT existente → Multidominio

↘ Exámen de CRT's

↘ Conversión de Key's y CRT's

BIBLIOGRAFIA:

→ OPENSSL COOKBOOK A Guide to the Most Frequently Used OpenSSL Features and Commands - Ivan Ristić (2016).

→ http://www.zytrax.com/tech/survival/ssl.html

→ https://es.wikipedia.org/wiki/X.509

→ https://wiki.openssl.org/index.php/Compilation_and_Installation

→ https://www.openssl.org/source/

→ https://es.wikipedia.org/wiki/RSA

→ https://es.wikipedia.org/wiki/DSA

→ https://es.wikipedia.org/wiki/ECDSA

→ https://es.wikipedia.org/wiki/Advanced_Encryption_Standard

→ https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations