

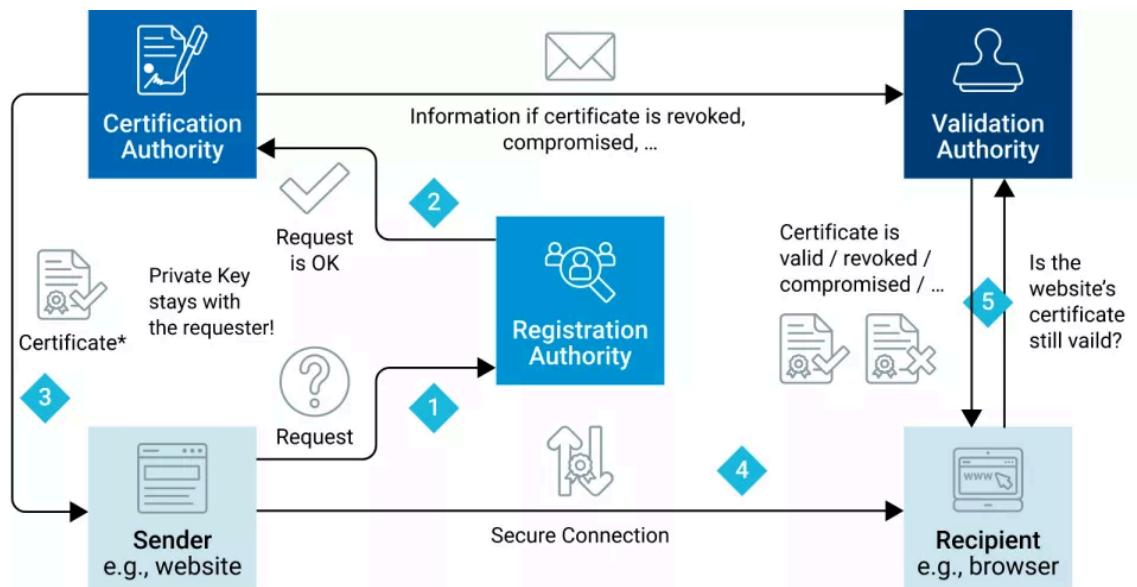
# SAD - U2.6 Certificado Digital

[Descargar estos apuntes](#)

## Índice

- ▼ 1. Certificados digitales
  - 1.1. Formato de un certificado digital
  
- ▼ 2. Certificados digitales X.509 con OpenSSL
  - 2.1. Ver un certificado
  - 2.2. Convertir entre formatos de certificados digitales
  - 2.3. Extraer claves desde un certificado
  
- ▼ 3. Creación de certificados digitales X.509 con OpenSSL
  - 3.1. Creación de un certificado para una CA (Certificado Raíz)
  - 3.2. Creación de un certificado firmado por una CA
  - 4. Conclusiones

## 1. Certificados digitales



\* The step of registering and requesting a certificate usually happens once each 1-2 years if it is a certificate used for HTTPS

Un **certificado digital x.509** es un documento digital que ha sido codificado y/o firmado digitalmente de acuerdo a la RFC 5280.

**x.509**

De hecho, el término «Certificado x.509» usualmente se refiere al certificado PKIX de la IETF, y al perfil CRL del estándar de certificados digitales x.509 v3 especificado en la RFC 5280, donde PKIX hace referencia a la Infraestructura de clave pública x.509 (Public Key Infrastructure X.509).

El estándar X.509 solo define la sintaxis de los certificados, por lo que no está atado a ningún algoritmo en particular, y contempla los siguientes campos en su estructura:

- **Version.**

Especifica cuál de las tres versiones de x.509 se aplica a este certificado.

- **Numero de serie.**

Es un identificador único para todos los certificados de una misma CA.

- **Identificador del algoritmo empleado para la firma digital.**

Qué algoritmo utilizó la CA del certificado para firmarlo.

- **Nombre del certificador.**

Nombre de la CA.

- **Periodo de validez.**

Por seguridad, los certificados tienen un tiempo de vida.

- **Nombre del sujeto.**

Nombre del dueño del certificado.

- **Clave publica del sujeto.**

Clave pública del dueño del certificado, y algoritmo asociado.

- **Extensiones.**

Información adicional permitida por el estándar.

- **Firma digital de todo lo anterior generada por el certificador.**

Firma digital de la CA.

Un ejemplo de certificado digital x.509 sería el siguiente:

**Certificate:****Data:**

Version: 3 (0x2)

Serial Number:

0a:01:41:42:00:00:01:53:85:73:6a:0b:85:ec:a7:08

Signature Algorithm: sha256WithRSAEncryption

Issuer: O = Digital Signature Trust Co., CN = DST Root CA X3

Validity

Not Before: Mar 17 16:40:46 2016 GMT

Not After : Mar 17 16:40:46 2021 GMT

Subject: C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

00:9c:d3:0c:f0:5a:e5:2e:47:b7:72:5d:37:83:b3:

[...]

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraints: critical

CA:TRUE, pathlen:0

X509v3 Key Usage: critical

Digital Signature, Certificate Sign, CRL Sign

Authority Information Access:

OCSP - URI:http://isrg.trustid.ocsp.identrust.com

CA Issuers - URI:http://apps.identrust.com/roots/dstrootcax3.p7c

X509v3 Authority Key Identifier:

C4:A7:B1:A4:7B:2C:71:FA:DB:E1:4B:90:75:FF:C4:15:60:85:89:10

X509v3 Certificate Policies:

Policy: 2.23.140.1.2.1

Policy: 1.3.6.1.4.1.44947.1.1.1

CPS: http://cps.root-x1.letsencrypt.org

X509v3 CRL Distribution Points:

Full Name:

URI:http://crl.identrust.com/DSTROOTCAX3CRL.crl

X509v3 Subject Key Identifier:

A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1

Signature Algorithm: sha256WithRSAEncryption

Signature Value:

dd:33:d7:11:f3:63:58:38:dd:18:15:fb:09:55:be:76:56:b9:

[...]

## 1.1. Formato de un certificado digital

Un punto importante a entender, y que puede generar confusiones entre los usuarios de certificados digitales X.509, son las extensiones (o formatos, mejor) que éstos pueden tener.

Existe mucha confusión al respecto de las extensiones **DER**, **PEM**, **CRT** y **CER**, y generalmente, y de manera incorrecta, muchos piensan que pueden utilizarse e intercambiarse sin generar conflictos.

Mientras que en ciertos casos algunas de estas extensiones pueden intercambiarse, la mejor práctica es la de identificar cómo está codificado internamente el certificado, y entonces etiquetarlo correctamente con la extensión que corresponda, ya que esto evitará problemas luego a la hora de manipular los certificados digitales.

## Codificación de los certificados digitales

Las siguientes codificaciones de formato también suelen utilizarse como extensiones en los archivos de certificados digitales. A saber:

- **DER**

La extensión DER es utilizada para certificados digitales codificados en forma binaria. Estos archivos también suelen tener la extensión CRT o CER. DER es un tipo de codificación de certificados X.509, NO un tipo de certificado, de modo que la expresión «Tengo un certificado DER» es incorrecta, y debería optarse por «Tengo un certificado X.509 codificado en formato DER».

- **PEM**

Estos certificados están codificados en Base64 / ASCII, y generalmente se utilizan para certificados digitales X.509 v3. Son aquellos que al abrirlos con un editor de texto comienzan y terminan con líneas similares a las siguientes:

```
-----BEGIN CERTIFICATE-----  
MIIDdzCCA1+gAwIBAgIJAJ7zU5z6g8IMA0GCSqGSIB3DQEBCwUAMIGUMQswCQYD  
VQQGEwJVUzETMBEGA1UECBMKU29tZS1TdGF0ZTEhMB8GA1UEChMYSW50ZXJuZXQg  
[...]  
-----END CERTIFICATE-----
```

Los archivos PEM pueden contener certificados, claves privadas, y otros datos. En el caso de los certificados digitales, la extensión PEM es la más común.

## Extensiones de certificados digitales X.509

- **CRT**

Esta extensión es utilizada para certificados propiamente dichos. Los certificados pueden ser codificados en binarios DER o como ASCII PEM. Las extensiones CER y CRT podrían considerarse sinónimos.

Es la extensión más utilizada en sistemas \*nix, como Linux o Unix.

- **CER**

Es una extensión alternativa a la CRT pero siguiendo las convenciones de Microsoft. La extensión CER es también reconocida por Internet Explorer como un comando para correr un ejecutable de la API de criptografía de Microsoft, y es utilizada principalmente por rundll32.exe, cryptext.dll, y CryptExtOpenCER, que muestran un mensaje de diálogo dando la posibilidad de importar el certificado en el sistema, o analizar su contenido.

La única ocasión en la que podemos intercambiar libremente las extensiones CRT y CER sin tener problemas, es cuando el certificado tiene la misma codificación... es decir, por ejemplo, cuando el certificado tiene extensión CRT o CER, pero su codificación es PEM.

- **KEY**

Esta extensión es utilizada tanto para la clave pública como la privada de los estándares PKCS. Estas claves pueden estar codificadas en formato binario DER o ASCII PEM.

## 2. Certificados digitales X.509 con OpenSSL

Vamos a analizar cuatro formas básicas de manipular certificados digitales X.509 utilizando openssl. Estas son, ver el certificado, convertirlo, combinarlo, y extraer claves desde el mismo.

Para proceder a gestionar los certificados se hará uso del comando `x509`. Este comando es una utilidad de OpenSSL que se utiliza para firmar, cifrar, descifrar, y verificar certificados digitales X.509.

## 2.1. Ver un certificado

Los certificados en general no tienen un formato legible al humano, incluso los formatos Base64 PEM, por lo que haremos uso de las herramientas provistas por OpenSSL para analizar su contenido.

Para ver el contenido de un certificado en formato PEM (independientemente de su extensión) podemos utilizar el siguiente comando:

```
$ openssl x509 -in /etc/ssl/cert.pem -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 6828503384748696800 (0x5ec3b7a6437fa4e0)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: CN = ACCVRAIZ1, OU = PKIACCV, O = ACCV, C = ES
    Validity
        Not Before: May 5 09:37:37 2011 GMT
        Not After : Dec 31 09:37:37 2030 GMT
    Subject: CN = ACCVRAIZ1, OU = PKIACCV, O = ACCV, C = ES
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
        Modulus:
            00:9b:a9:ab:bf:61:4a:97:af:2f:97:66:9a:74:5f:
            [...]
        Exponent: 65537 (0x10001)
X509v3 extensions:
    Authority Information Access:
        CA Issuers - URI:http://www.accv.es/fileadmin/Archivos/certificados/raizacccv1.crt
        OCSP - URI:http://ocsp.accv.es
    X509v3 Subject Key Identifier:
        D2:87:B4:E3:DF:37:27:93:55:F6:56:EA:81:E5:36:CC:8C:1E:3F:BD
    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Authority Key Identifier:
        D2:87:B4:E3:DF:37:27:93:55:F6:56:EA:81:E5:36:CC:8C:1E:3F:BD
    X509v3 Certificate Policies:
        Policy: X509v3 Any Policy
        User Notice:
            Explicit Text:
            CPS: http://www.accv.es/legislacion_c.htm
    X509v3 CRL Distribution Points:
        Full Name:
            URI:http://www.accv.es/fileadmin/Archivos/certificados/raizacccv1_der.crl
    X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    X509v3 Subject Alternative Name:
        email:acccv@acccv.es
Signature Algorithm: sha1WithRSAEncryption
Signature Value:
97:31:02:9f:e7:fd:43:67:48:44:14:e4:29:87:ed:4c:28:66:
[...]
```

### ⚠️ Error en el formato del certificado

En el caso de obtener un error similar a este:

```
unable to load certificate  
12626:error:0906D06C:PEM routines:PEM_read_bio:no start line:pem_lib.c:647:Expecting: TRUSTED CERTIFICATE
```

Es posible que el archivo no sea un certificado en formato PEM, sino en formato DER. En este caso, se debe convertir el certificado a formato PEM antes de poder visualizarlo. También podemos usar un comando similar a este para visualizar certificados en formato DER:

```
$ openssl x509 -in certificado.der -inform der -text -noout
```

## 2.2. Convertir entre formatos de certificados digitales

A veces, y dependiendo de las aplicaciones en las que vayamos a utilizar los certificados, puede que sea necesario cambiar el formato de un certificado.

Esto no es tarea difícil para la suite OpenSSL, y podemos llevarla a cabo de las siguientes maneras:

Convirtiendo PEM a DER:

```
openssl x509 -in cert.crt -outform der -out cert.der
```

Convirtiendo DER a PEM:

```
openssl x509 -in cert.crt -inform der -outform pem -out cert.pem
```

## 2.3. Extraer claves desde un certificado

Por último, y sabiendo que un certificado digital X.509 se compone de metainformación agregada a una clave pública, utilizando la suite OpenSSL también podemos extraer la clave desde el certificado.

Supongamos que tenemos un certificado en formato PEM (si es DER, podemos convertirlo como se vio mas arriba). Podemos extraer la clave pública con un comando similar a este:

```
openssl x509 -inform pem -in cert.crt -pubkey -noout > pub.key
```

Donde pub.key es la clave pública resultante, también en formato PEM.

## 3. Creación de certificados digitales X.509 con OpenSSL

La criptografía asimétrica se basa en dos claves, una pública y una privada. Cuando hablamos de PKI y mencionamos que lo que utilizan los nodos de una comunicación segura son certificados digitales, cabe resaltar que **un certificado digital es la clave pública de una entidad, a la que se ha añadido información de validez (hashes, firmas digitales, fechas de expiración, etc)**.

### 3.1. Creación de un certificado para una CA (Certificado Raíz)

#### ⚠ Certificados autofirmados

No tenemos todavía una autoridad certificante, o **CA**, por lo que vamos a crear nuestro primer certificado firmado por la misma clave privada asociada a la clave pública incluida en el certificado.

A esto se lo denomina certificado autofirmado.

- Primero creamos nuestra clave privada RSA, privateRSAkey.pem.

```
$ openssl genpkey -algorithm RSA -out privateRSA.key
```

- En segundo lugar, generamos la petición de firma de certificado, o **CSR**, que idealmente utilizará una autoridad certificante para darnos un certificado firmado que incluya nuestra clave pública.

El archivo CSR (Certificate Sign Request), en nuestro caso inicial, vamos a firmarlo nosotros mismos con la clave privada que acabamos de crear.

```
$ openssl req -new -key privateRSA.key -out certRSA.csr
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.
-----
Organization Name (company) [My Company]:IES Doctor Balmis
Organizational Unit Name (department, division) []:Dpto. Informática
Email Address []:vicente@iesdoctorbalmis.com
Locality Name (city, district) []:Alicante
State or Province Name (full name) []:Alicante
Country Name (2 letter code) []:es
Common Name (hostname, IP, or your name) []:Vicente Martínez Martínez
```

- En tercer lugar, vamos a generar, a partir del CSR y de la clave privada, un certificado x509 autofirmado.

Carguemos, por ejemplo, un certificado válido por 365 días.

```
$ openssl x509 -req -days 365 -in certRSA.csr -signkey privateRSA.key -out certificadoRaiz.crt
Certificate request self-signature ok
subject=O=IES Doctor Balmis, OU=Dpto. Informática, emailAddress=vicente@iesdoctorbalmis.com, L=Alicante, ST=Alicante
```

- Ya por último, verificamos que el certificado autofirmado se ha creado correctamente.

Primero habrá que introducir el certificado de la autoridad certificadora, y después nuestro certificado, que en nuestro caso es el mismo certificado raíz.

Después podemos ver el contenido del certificado.

```
$ openssl verify -CAfile certificadoRaiz.crt certificadoRaiz.crt
certRSA.crt: OK
$ openssl x509 -in certificadoRaiz.crt -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        63:29:f5:bf:a7:81:05:76:4c:84:20:3b:38:ac:48:cd:c2:84:38:96
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: O=IES Doctor Balmis, OU=Dpto. Informática, emailAddress=vicente@iesdoctorbalmis.com, L=Alicante, ST=Spain
    Validity
        Not Before: Oct 17 21:31:36 2024 GMT
        Not After : Oct 13 21:31:36 2025 GMT
    Subject: O=IES Doctor Balmis, OU=Dpto. Informática, emailAddress=vicente@iesdoctorbalmis.com, L=Alicante, ST=Spain
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:99:5c:3b:6d:38:d2:bf:e1:ac:c7:5b:4f:56:ab:
            2c:00:b6:af:73:8b:4c:fc:57:12:3f:35:77:d7:f3:
            d7:fd:60:4a:ea:4f:22:e3:4d:33:49:35:92:d2:69:
            b6:ce:d4:eb:01:80:8c:35:55:79:66:db:9e:de:87:
            74:d0:ee:3f:68:9e:2d:d3:3b:45:4e:97:1f:67:9d:
            c9:32:f8:5d:35:ac:08:4a:c5:9c:b1:53:e5:74:44:
            fe:22:52:b5:02:4d:70:66:0f:87:59:0b:58:f4:ed:
            63:8b:a6:f8:43:34:1b:9f:4d:45:23:a0:c4:40:67:
            21:f1:19:c7:f1:5a:18:7e:f5:b0:86:11:a4:b9:b2:
            e8:7c:6a:9b:e7:6f:fa:93:ee:95:20:13:85:fa:e1:
            d5:3e:96:4f:b3:09:1d:b9:4d:4c:26:30:9e:e6:79:
            b4:d6:33:c8:6d:96:41:91:aa:49:80:b8:d6:d8:fd:
            b1:f1:f6:14:0b:be:b8:c2:40:9f:cc:50:2b:a8:0d:
            80:50:24:55:74:ca:26:5b:37:a7:36:c6:da:18:72:
            20:9a:8c:3e:08:02:5b:37:aa:ed:fe:69:2f:56:02:
            e4:c5:46:e0:71:78:23:f4:73:4c:7f:32:71:9a:aa:
            87:9b:70:7a:32:87:15:ba:8c:02:3c:76:8e:7c:f8:
            63:d9
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        7B:BD:31:99:D1:04:99:79:8B:AD:C1:29:C9:0A:88:3F:35:88:B0:A5
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
6d:5e:ab:0e:d0:b1:d1:06:0d:70:18:26:14:2e:dc:21:1a:bc:
ee:19:98:52:15:03:fb:ce:98:a0:44:1d:7f:03:ca:54:da:ef:
e8:d3:78:f7:6d:8f:57:45:0d:bf:74:7b:64:a7:1d:65:e8:55:
ea:8f:7a:70:44:78:f8:07:9a:37:7f:bb:66:02:3b:12:e4:f0:
71:60:d8:7c:1c:a2:86:a8:37:79:c7:21:1f:6d:f4:5d:9e:5f:
b4:95:68:1f:10:87:23:a9:06:3e:2b:46:b0:f3:9a:2b:13:f0:
05:e2:58:3c:5b:31:88:4c:ee:93:b9:5e:5c:f1:9f:49:31:f5:
2e:e7:04:8b:0d:74:90:03:b6:b8:44:ca:d4:ab:ce:1d:0b:cc:
88:76:52:4a:11:e3:86:1f:1b:76:87:36:67:0f:6d:06:8b:a5:
b3:d0:d1:c7:f8:8f:09:50:48:f8:b6:79:3b:f3:ed:2a:8b:b7:
30:25:26:da:2b:2d:f7:9a:ae:b5:6e:b3:f6:04:73:61:28:20:
e8:9c:8f:27:aa:3a:9d:f5:ca:39:8e:f3:c8:f3:ee:f2:53:c0:
```

```
6a:05:35:29:83:6a:91:aa:83:4e:7f:96:50:ff:3c:7c:0b:bb:  
94:79:ca:70:e3:23:25:d9:62:96:75:de:f1:da:a3:32:8e:e8:  
18:71:bf:ae
```

### Certificado no confiable

Si importamos el certificado en un navegador, por ejemplo, nos dirá que no es confiable.

Si tuviésemos la clave privada "CA.key" de alguna autoridad de confianza, se podría usar a la hora de firmar el nuevo Certificado creado por nosotros.

Así obtendríamos un certificado válido firmado con la clave privada de dicha autoridad.

OpenSSL provee una forma de crear un certificado digital x509 autofirmado y su clave privada en un solo comando, de manera muy sencilla:

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.

Organization Name (company) [My Company]:IES Doctor Balmis  
Organizational Unit Name (department, division) []:Dpto. Informática  
Email Address []:vicente@iesdoctorbalmis.com  
Locality Name (city, district) []:Alicante  
State or Province Name (full name) []:Alicante  
Country Name (2 letter code) []:es  
Common Name (hostname, IP, or your name) []:Vicente Martínez Martínez

### **3.2. Creación de un certificado firmado por una CA**

Ahora que ya tenemos una CA, aunque la hayamos creado nosotros, vamos a generar un certificado digital firmado por la CA creada por nosotros mismos.

Los pasos son similares a los anteriores, pero en este caso vamos a firmar el CSR con la clave privada de la CA.

Vamos a crear un certificado que podemos usar en un servidor web, por ejemplo.

- Primero creamos la clave privada RSA para el servidor web.

```
$ openssl genpkey -algorithm RSA -out privateWebServerRSA.key
```

- En segundo lugar, generamos la petición de firma de certificado para el servidor web.

```
$ openssl req -new -key privateWebServerRSA.key -out certWebServer.csr
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.
-----
Organization Name (company) [My Company]:IES Doctor Balmis
Organizational Unit Name (department, division) []:Dpto. Informática
Email Address []:vicente@iesdoctorbalmis.com
Locality Name (city, district) []:Alicante
State or Province Name (full name) []:Alicante
Country Name (2 letter code) []:es
Common Name (hostname, IP, or your name) []:www.sad.asir.iesdoctorbalmis.com
```

- En tercer y último lugar, vamos a generar, a partir del CSR y de la clave privada, un certificado x509 firmado por la CA.

```
$ openssl x509 -req -days 365 -sha512 -in certWebServer.csr -CA certificadoRaiz.crt -CAkey privateRSA.key -CAcreateserial
Certificate request self-signature ok
subject=O=IES Doctor Balmis, OU=Dpto. Informática, emailAddress=vicente@iesdoctorbalmis.com, L=Alicante, ST=Alicante
```

En este caso he especificado un hash SHA512 para la generación de la firma. Recuerden **no usar nada menor a SHA256 por seguridad.**

Cuando se emite un certificado mediante una Autoridad de Certificación (CA), se debe asignar un **número de serie único a cada certificado** emitido. Este número de serie se incluye en el certificado y se utiliza para identificarlo de manera única. Cuando se trabaja con CAs, algunas versiones de openssl requieren que se cree un archivo donde se almacene el número de serie de los certificados firmados.

La opción `-CAcreateserial` se utiliza cuando se genera un nuevo certificado firmado por una CA existente. Si el archivo de serie aún no existe, esta opción creará un nuevo archivo y asignará el número de serie 01 al primer certificado emitido. Si el archivo ya existe, se continuará utilizando el último número de serie disponible en el archivo existente.

Y sí, podemos verificar la firma si quisiéramos, pero para que la verificación sea exitosa, deberíamos añadir el certificado de la CA a la lista de certificados de confianza del sistema.

```
$ cat certificadoRaiz.crt certWebServer.crt > certWebServerCA.crt

$ openssl verify -CAfile certificadoRaiz.crt certWebServerCa.crt
cert_sitio.crt: OK
```

## 4. Conclusiones

En la vida real no usamos CAs nuestras CA, sino que utilizamos CAs reconocidas por los navegadores, como Verisign, Thawte, etc. Estas CAs tienen sus propias CAs que les certifican a ellas, y así sucesivamente.

Cuando montamos un servidor en producción, lo más recomendable es que no creemos nuestra propia CA para firmar los certificados, sino que creemos la petición de firma (el CSR), y le solicitemos a una autoridad certificante válida la generación

del certificado firmado. Estas CAs son conocidas, y los navegadores confían en ellas (es decir, ya disponen de su certificado para verificar la firma).

Algunas CAs conocidas son las siguientes:

- Comodo
- GeoTrust
- GoDaddy
- DigiCert

Todas estas CAs son de pago, los precios, para tener una idea, rondan los **400 dólares por año por dominio**.

No obstante, una autoridad certificante válida gratuita muy utilizada es **Let's Encrypt**, una organización sin fines de lucro que se encarga de firmar certificados por períodos más cortos de tiempo, y dispone de herramientas para la actualización automática de certificados digitales.

Para acabar, podemos preguntarnos:

¿Cómo va a confiar el navegador del cliente en la CA creada por nosotros?

La única forma es compartirle el certificado de la CA (el que usamos para verificar la firma), de modo que el cliente lo pueda añadir a la lista de certificados válidos en su navegador.

Si generamos certificados para uso dentro de una empresa, podemos añadir el certificado de la CA a la lista de certificados de confianza de los equipos de la empresa cuando hacemos la instalación de los equipos, y así no tendremos problemas de confianza.