Seguridad en Redes de Comunicación

Práctica II. Despliegue de una PKI

Jorge Lanza Calderón

Luis Sánchez González

Departamento de Ingeniería de Comunicaciones

Práctica 2

Despliegue de una PKI

1. Objetivo de la práctica

El objetivo de esta práctica es llevar a cabo el despliegue de una infraestructura de clave pública (PKI, *Public Key Infrastructure*) y realizar sobre ella las operaciones administrativas y de uso más comunes.

- Creación y configuración de una autoridad de certificación (CA, *Certification Authority*).
- Generación y firma de certificados.
- Operativa con certificados.
- Revocación de certificados basada en listas de revocación (CRL, *Certificate Revocation List*).
- Configuración y uso de un servidor OCSP (*Online Certificate Status Protocol*).
- Aplicación de sellado de tiempo (TSA, *Time Stamping Authority*).

2. Introducción

El principal problema de los criptosistemas de clave pública reside en garantizar la propiedad y la validez de las claves públicas. Se considera un certificado como la asociación del nombre de una entidad con su clave pública durante un periodo de validez avalada por una tercera entidad de confianza.

Una PKI es el conjunto de dispositivos, aplicaciones, personas, políticas y procedimientos que son necesarios para crear, administrar, distribuir y revocar certificados basados en criptografía de clave pública.

Las diferentes entidades que conforman una PKI permiten asegurar la cadena de confianza a la hora de establecer relaciones seguras entre dos usuarios, sus máquinas, etc. En una PKI se pueden distinguir los siguientes componentes:

- Autoridad de certificación (CA): para generar los certificados.
- Autoridad de registro (RA): para la identificación y el registro previo de identidades antes de ser emitidos los certificados.
- Repositorios o directorios: para el almacenamiento y recuperación de certificados y listas de revocación (CRL).
- Autoridad para la generación y distribución de CRL.

- Autoridad de sellado de tiempos (TSA): para dar fe de la existencia de una determinada información en un momento concreto.

El formato universalmente aceptado para definir un certificado es X.509. Este formato surgió en 1988 y actualmente se encuentra, en su versión 3, descrito en el RFC 5280. Se utiliza en multitud de aplicaciones como S/MIME, IPSec, SSL/TLS, etc.

3. Entorno de desarrollo

La práctica se desarrolla en una máquina ejecutando el sistema operativo *Ubuntu*. Se utilizará el usuario `alumnos`, con contraseña `telematica`. Debido a la configuración del entorno en el que se realiza la práctica, los alumnos deben guardar los archivos en los que hayan trabajado para no perder los cambios, y poder empezar con ellos durante la sesión siguiente.

A lo largo de esta práctica se emplearán las herramientas implementadas por OpenSSL a través de la línea de comandos usando el programa `openssl`.

```
$ man openssl
```

4. Desarrollo

La práctica plantea el despliegue de una PKI en un entorno local basada en una jerarquía de confianza de un único nivel. Para ello, a continuación, se detallan los procedimientos de configuración del sistema para poder alojar las diferentes entidades de la PKI, así como la secuencia de pasos necesarios para generarlas. También se describen los mecanismos de expedición, publicación y validación certificados.

4.1. Configuración y creación de una CA

Aunque el comando `openssl` permite la ejecución de diferentes opciones directamente a través de argumentos de programa, en numerosas ocasiones y dependiendo del comando, muchos de los parámetros pueden tomar valores por defecto según lo definido en un fichero de configuración.

```
$ man config
$ openssl <command> -config <openssl.cnf> <command_arguments>
$ export OPENSSL_CONF=<path_to_configuration_file>
```

Antes de proceder con la propia configuración de la CA se tiene que definir la estructura de ficheros/directorios donde se alojarán los directorios de certificados.

```
$ mkdir ca
$ cd ca
$ mkdir certs
$ mkdir crl
$ mkdir private
```

El directorio `private` es el lugar donde se almacenarán las claves privadas necesarias. Se recomienda protegerlo de forma que sólo pueda ser accesible para el dueño de la CA.

```
$ chmod 700 private
```

Definida la estructura de la CA, el siguiente paso es realizar su configuración. Para facilitar la tarea en esta práctica se genera un fichero de configuración en el que se incluirán las opciones adecuadas. Un ejemplo de dicho fichero, que se puede tomar como base, se muestra en el apéndice A, aunque también es válido tomar como referencia el fichero incluido en el sistema.

```
$ locate openssl.cnf
$ less /etc/ssl/openssl.cnf
```

El formato del fichero de configuración de OpenSSL se asemeja al formato de un fichero INI. Este fichero incluye diversas secciones donde se asignan los parámetros para acceder a los directorios de certificados, así como los valores que se asociarán a los diferentes campos y extensiones que se incluirán en los certificados.

```
$ man config
$ man x509v3_config
```

Cuestión I

1) Sabiendo que la sección `[ ca ]` (redirigida a `[ CA_default ]`) establece el emplazamiento del directorio de certificados, se pide actualizar su contenido para adecuarlo al caso específico de cada usuario.

4.2. Generación de la clave y certificado de la CA

Se procede seguidamente a generar la clave privada y el certificado de la CA raíz de la jerarquía de confianza.

Una posibilidad es crear tanto la clave privada como la petición de firma de certificado (CSR, *Certificate Signing Request*) en el mismo paso. A la hora de generar una CSR el valor de los parámetros asignados por defecto se encuentra en la sección `[ req ]` del fichero de configuración que se emplee. Rellenar los diferentes campos del DN (Distinguished Name) teniendo en cuenta las políticas que se establecen para su validación a la hora de ser firmado por la CA.

```
$ openssl req -new -newkey rsa:2048 -keyout private/ca.key.pem \
  -out ca.csr.pem
$ openssl req -in ca.csr.pem -text -noout
```

No obstante, puede ocurrir que se tenga una clave privada y se desee generar la CSR a partir de dicha clave[1].

```
$ openssl req -new -key private/ca.key.pem -out ca.csr.pem
```

La CA que se está generando es una CA raíz y el certificado de ella se plantea por tanto autofirmado. Queda definir tanto la duración temporal como las extensiones a aplicar al certificado que se genere.

[1] La generación de una pareja de claves pública y privada se ha tratado en la Práctica 1

Cuestión II

1) En el fichero `openssl.cnf` se incluirá una sección `[ v3_ca ]` en la que se especificarán las extensiones `basicConstraints`, `subjectKeyIdentifier` y `authorityKeyIdentifier`. Se pide que dichas secciones se ajusten para definir una CA que no permita la definición de certificados de otras CA.

Tras realizar la configuración deseada se procede a generar el certificado de la CA estableciendo una duración de 20 años.

```
$ openssl x509 -req -days 7300 -in ca.csr.pem -signkey private/ca.key.pem \
  -extfile ./openssl.cnf -extensions v3_ca -out ca.crt.pem
```

Se podría haber realizado todo el procedimiento ejecutando una única instrucción sin tener acceso a la CSR de la CA y generando un número de serie aleatorio, pero se recomienda seguir el proceso anterior para tener mejor conocimiento de los pasos.

```
$ openssl req -new -x509 -key private/ca.key.pem -out ca.crt.pem \
  -days 7300 -extensions v3_ca
```

Una vez que se tenga el certificado de la CA, generar los ficheros que alojan la base de datos de certificados y el próximo número de serie de certificado generado.

```
$ touch index.txt
$ echo 01 > serial
```

Nuevamente existe la posibilidad de realizar todo el proceso de forma directa mediante el comando `ca` de `openssl`.

```
$ touch index.txt
$ openssl ca -create_serial -out ca.crt.pem -days 7300 \
  -keyfile private/ca.key.pem -selfsign -extensions v3_ca \
  -infiles ca.csr.pem
```

Con este comando se genera la clave privada y el certificado autofirmado de una CA, además de generar el fichero `serial` con un contenido aleatorio. Además, el certificado asignado a la CA queda registrado como el primer certificado de la CA, como se puede comprobar visualizando el contenido de `index.txt`.

Cuestión III

1) Describir las diferencias que se observan entre la CSR y el certificado de la CA.

   ```
   $ openssl x509 -in <x509doc> -text -noout
   ```

   Si se ha optado por generar el certificado autofirmado de la CA de manera directa, se puede extraer la CSR mediante el siguiente comando:

   ```
   $ openssl x509 -x509toreq -in ca.crt.pem -out ca.csr.pem \
     -signkey private/ca.key.pem
   ```

4.3. Generación de certificados

Una vez que se ha configurado la CA, se procede a generar los certificados de usuario y/o servidor.

Aunque de manera opcional, es recomendable que un certificado X.509v3 de usuario final incluya una serie de extensiones.

```
[ usr_cert ]
...
basicConstraints=CA:FALSE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
subjectAltName=email:move
...
```

Un certificado de usuario que se pueda emplear en la autenticación en entornos web seguros basados en SSL (*Secure Socket Layer*) debe incluir valores específicos en las extensiones relacionadas con el uso de la clave asociada al certificado. Para ello se actualiza el fichero de configuración con una nueva sección.

```
[ usr_cert ]
...
keyUsage = digitalSignature,nonRepudiation,keyAgreement
extendedKeyUsage = clientAuth
...
```

Adicionalmente se podrán incluir extensiones de carácter informativo acerca de la localización de las opciones de validación (puntos de distribución de las CRL y/o servidor OCSP) y de la entidad emisora. Se incluyen a continuación algunos posibles valores, a modo de ejemplo, para comprender el formato. Modificarlos para ajustarse a la configuración específica de cada PKI.

```
[ usr_cert ]
...
crlDistributionPoints=URI:http://myserver.com/testlabcrl.crt
authorityInfoAccess=caIssuers;URI:http://www.domain.com/ca.html,OCSP;URI:
  http://ocsp.domain.com/
...
```

Actualizado el fichero de configuración con la sección que engloba las opciones para el certificado de usuario, se puede proceder a generar dicho certificado.

```
$ openssl req -newkey rsa:2048 -keyout <client_private_key> -nodes \
  -out <client_csr>
$ openssl ca -in <client_csr> -extensions usr_cert -out <client_cert> \
  -notext
$ openssl x509 -in <client_crt> -text -noout
```

Cuestión IV

1) Generar un certificado de servidor, adecuando el contenido de las extensiones para que pueda ser empleado en la autenticación y establecimiento de la sesión segura.

2) ¿Qué ocurre si se modifica el país o la provincia y no se acepta el valor por defecto a la hora de introducir los datos del usuario? Razonar la respuesta.

3) Una vez que se hayan generado varios certificados, ¿cuál es el contenido del directorio `certs` y qué relación tiene con el contenido del fichero `index.txt`?

4.4. Operativa con certificados

4.4.1. Formatos de almacenamiento

Los certificados se pueden almacenar en muy diversos formatos (PEM, DER, PFX, etc.). A continuación se presentan algunas opciones para transformar de uno a otro formato contenedor.

```
$ echo "Convert PEM to DER"
$ openssl x509 -outform der -in certificate.pem -out certificate.der
$ echo "Convert PEM to P7B"
$ openssl crl2pkcs7 -nocrl -certfile certificate.cer -out certificate.p7b \
  -certfile CACert.cer
$ echo "Convert PEM to PFX"
$ openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key \
  -in certificate.crt -certfile CACert.crt

$ echo "Convert P7B to PEM"
$ $openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
$ echo "Convert P7B to PFX"
$ openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
$ openssl pkcs12 -export -in certificate.cer -inkey privateKey.key \
  -out certificate.pfx -certfile CACert.cer

$ echo "Convert PFX to PEM"
$ openssl pkcs12 -in certificate.pfx -out certificate.cer -nodes
```

4.4.2. Verificación

Se verifica a continuación de forma local que los certificados recién creados están correctamente construidos y firmados por la CA. Si se modificara el contenido del certificado la verificación no sería exitosa.

```
$ openssl verify -CAfile ca.crt.pem <client_cert>
```

El argumento `CAfile` incluye un fichero con toda la cadena de certificados que sirva para verificar el certificado[2]. En este caso, dado que el certificado de la CA es autofirmado solo

[2]La opción `-CApath` implementa la misma funcionalidad, pero los certificados de confianza se encuentran en un directorio

contendrá un certificado.

```
$ cat ca_root.pem ca_1.pem ca_2.pem ca_3.pem ... > ca_bundle.crt
```

De forma local también se puede verificar la validez temporal de un certificado. `openssl` no ofrece ninguna opción para comprobar de forma automática que la fecha actual se encuentra en el rango de fechas incluidas en el certificado.

```
$ openssl x509 -in <client_cert> -noout -dates
```

4.4.3. Firma de documentos

Generar un fichero de un tamaño superior a 3Kb.

```
$ dd if=/dev/urandom of=file.bin bs=3000 count=1
```

Realizar la firma. Existen al menos dos posibilidades de realizar este procedimiento.

```
$ openssl dgst -sha1 -binary -sign <client_key> -out <signature> \
  <file_to_sign>

$ openssl dgst -sha1 -binary <file_to_hash> > <file_hash>
$ openssl pkeyutl -sign -inkey <client_key> -in <file_hash> \
  -pkeyopt digest:<alg> -out <signature>
```

Y verificar posteriormente la correcta generación de la firma. Modificar el fichero origen o el fichero de firma y realizar el procedimiento de verificación.

```
$ openssl x509 -in <client_certificate> -pubkey -noout > <client_public_key>
$ openssl dgst -<alg> -verify <client_public_key> -signature <signature> \
  <received_file>

$ openssl pkeyutl -verify -certin -inkey <client_certificate> \
  -sigfile <signature> -in <file_hash> -pkeyopt digest:<alg>
$ openssl pkeyutl -verify -pubin -inkey <client_public_key> \
  -sigfile <signature> -in <file_hash> -pkeyopt digest:<alg>
```

Los procedimientos realizados complementan a los introducidos en la Práctica 1, siendo en este caso la clave pública distribuida mediante un certificado.

4.5. Revocación de certificados

La información vinculada a un certificado puede verse comprometida, lo que supone que la confianza de la entidad emisora se extingue. Las CRL son uno de los métodos a través de los cuales la CA emisora publicita qué certificados han dejado de tener validez. Usualmente como se ha visto se incluyen los puntos de distribución de CRL entre las extensiones de un certificado. A continuación se describe el procedimiento para revocar certificados, generar CRL y verificar la validez de un certificado.

Primeramente se debe configurar el directorio de certificados para soportar CRL.

```
$ echo 01 > crlnumber
```

Actualizar el fichero de configuración de OpenSSL para habilitar el uso de CRL versión 2 e incluir el conjunto de extensiones deseadas.

```
[ crl_ext ]
...
authorityKeyIdentifier=keyid:always,issuer:always
...
```

Generar una CRL y analizar su contenido comprobando que se ajusta a la configuración establecida.

```
$ openssl ca -keyfile <ca_private_key> -cert <ca_cert> -gencrl \
  -crldays 3 -crlexts crl_ext -out crl/crl`cat crlnumber`.pem
$ openssl crl -in crl/crl<number>.pem -text -noout
```

Seguidamente, revocar cualquier de los certificados anteriormente generados.

```
$ openssl ca -crl_reason <reason> -revoke <client_cert>
$ openssl ca -keyfile <ca_private_key> -cert <ca_cert> -gencrl \
  -crldays 3 -crlexts crl_ext -out crl/crl`cat crlnumber`.pem
$ openssl crl -in crl/crl<number>.pem -text -noout
```

Cuestión V

1) Analizar el contenido de la base de datos de certificados (fichero `index.txt`) y determinar la diferencia entre los diferentes certificados en ella listada. ¿Se corresponde la información con la incluida en la CRL generada?

2) ¿Se observa alguna diferencia entre el contenido de un certificado revocado y otro válido?

3) En los certificados se puede incluir la extensión `crlDistributionPoints` que informa de las localizaciones donde está disponible la CRL. ¿Es necesario que la URI (*Uniform Resource Identifier*) que se establece implique un acceso sobre un canal seguro o empleando un protocolo seguro (i.e. HTTPS (*HyperText Transfer Protocol Secure*))? Razonar la respuesta.

4) Renovar uno de los certificados anteriormente revocados.

Según se informa en el manual de OpenSSL, la creación y gestión de delta-CRL no están aún implementadas, motivo por el cual no se podrá evaluar su funcionalidad en esta práctica.

4.6. Validación de certificados

4.6.1. Basada en CRL

Proceder a continuación a verificar el estado de un certificado de forma automática. Para ello generar la cadena de confianza de la CRL que luego se emplea en el proceso de verificación. Esta información se hace pública a través de la información disponible en las extensiones `authorityInfoAccess` y `crlDistributionPoints`.

Comprobar de manera local el estado de varios de los certificados emitidos y validar que se corresponde con lo obtenido a través de la inspección de la CRL y de la base de datos.

```
$ cat <ca_cert> crl.pem > revoke-test.pem
$ openssl verify -CAfile revoke-test.pem -crl_check <certificate_to_check>
```

Para habilitar el acceso remoto a la CRL y el certificado de la CA se requiere la instalación de un servidor web. Sustentado en la configuración por defecto del servidor Apache se muestra un procedimiento básico para publicar la CRL, suponiendo que las extensiones `authorityInfoAccess` y `crlDistributionPoints` tienen asignado correctamente los valores asociados al sistema (direcciones, etc.).

```
$ sudo apt-get install apache2
$ cd /var/www
$ sudo mkdir SRC
$ cd SRC
$ sudo ln -s <path_to_crl> <my_crl_name>.crl
$ sudo ln -s <path_to_ca> <my_ca_name>.crt
$ wget http://my.domanin.com/<my_crl_name>.crl
$ wget http://my.domanin.com/<my_ca_name>.crt
```

Cuestión VI

1) Enviar a un compañero un mensaje firmado junto con el certificado asociado y habilitar la verificación de dicho certificado a través de CRL. Puede ser necesario modificar la configuración realizada en el apartado 4.3.

4.6.2. Basada en OCSP

OCSP es un protocolo diseñado para realizar la validación de certificados X.509 en tiempo real. OCSP intercambia mensajes codificados en ASN.1 y usualmente sobre HTTP.

Primeramente se procederá a realizar la validación de un certificado en un entorno de producción, para posteriormente desplegar un servidor OCSP propio. En ambos casos se realizará la captura de las transmisiones y se analizará su contenido mediante `Wireshark`.

4.6.2.1. Servidor comercial

Se usará la validación OCSP para comprobar el estado de los certificados empleados en el acceso seguro a la web de la Universidad de Cantabria (`https://www.unican.es`).

Este sitio web emplea un certificado cuya cadena de confianza tiene varios niveles.

```
$ openssl s_client -showcerts -connect www.unican.es:443 < /dev/null | \
awk -v c=-1 '/-----BEGIN CERTIFICATE-----/{inc=1;c++}
            inc {print > ("level" c ".crt")}
            /---END CERTIFICATE-----/{inc=0}'
$ ls level*.crt
```

Cada uno de estos certificados se corresponde con un nivel diferente. Identificar la entidad final, el número de serie, el emisor y el periodo de validez de cada uno de ellos y obtener la dirección del servidor OCSP.

```
$ openssl x509 -noout -serial -subject -issuer -dates -in levelXX.crt
$ openssl x509 -noout -text -in levelXX.crt | grep OCSP
```

Antes de realizar la validación de los certificados y dado que la respuesta del servidor OCSP normalmente viene firmada se requiere confeccionar un lista de certificados de confianza. Se emplearán los certificados de la jerarquía del servidor al que se ha accedido, a los que se pueden incorporar aquellos incluidos por defecto en el sistema.

```
$ cat /etc/ssl/certs/ca-certificates.crt level{1,2}.crt > CAbundle.crt
```

Realizar la validación accediendo al servidor OCSP apropiado dependiendo del certificado que se quiera validar.

```
$ openssl ocsp -issuer level1.crt -nonce -url <oscp_server_url> \
  -cert level0.crt -CAfile CAbundle.crt -VAfile CAbundle.crt -text

$ openssl ocsp -issuer level1.crt -nonce -url <oscp_server_url> \
  -serial <serial_level0> -CAfile CAbundle.crt -VAfile CAbundle.crt -text
```

Algunos servidores OCSP basado en servidores web que operan empleando el protocolo HTTP 1.1 (i.e. servidor OCSP de Gmail). En esos casos es necesario incluir un parámetro adicional en la petición de validación.

```
$ openssl ocsp -issuer level1.crt -nonce -url <oscp_server_url> \
  -cert level0.crt -CAfile CAbundle.crt -VAfile CAbundle.crt -header "HOST"
    "ocsp_server_domain_name" -text
```

4.6.2.2. Servidor propio

Para desplegar un servidor OCSP propio se emplean los ficheros que dan soporte al directorio de certificados.

Cuestión VII

1) Crear la clave privada y el certificado para el servidor OCSP a partir de los cuales se firmarán las respuestas. Adecuar según corresponda las extensiones de uso de clave.

OpenSSL permite ejecutar un servidor OCSP de una manera sencilla. Se recomienda usar el puerto 8080 para evitar colisiones con cualquier otro servidor que esté ejecutándose en el sistema.

```
$ openssl ocsp -index index.txt -CA <ca_cert> -rsigner <ocsp_cert> \
  -rkey <ocsp_key> -port <port>
```

Verificar el estado de cualquiera de los certificados emitidos. Evaluar la operativa de forma remota, es decir, indicando a algún compañero los parámetros del servidor desplegado y de alguno de los certificados (se podrá emplear el certificado enviado anteriormente).

```
$ openssl ocsp -issuer <issuer_cert> -nonce -url http://<host>:<port> \
 -cert <cert_to_check> -CA <ca_cert> -VAfile <ocsp_cert> -text
```

Cuestión VIII

1) ¿Se ha observado alguna diferencia entre la respuestas obtenidas en el servidor propio y el servidor desplegado en producción?

4.7. Sellado de tiempo

La TSA actúa como una tercera parte de confianza que garantiza de forma objetiva y precisa, a través de la entrega de Sellos de Tiempo, la existencia de una información en un instante de tiempo determinado.

4.7.1. TSA propia

Definir la configuración de la TSA en el fichero de configuración. Del mismo modo que se realizó para la CA y la CRL, se fijan las localizaciones de los ficheros que contienen la clave privada y el certificado de la TSA, el número de serie de los sellos emitidos y características adicionales de las respuestas que se generarán a partir de las peticiones de sellado de tiempo.

Adicionalmente es necesario definir las extensiones que debe incluir el certificado asociado a la TSA. Prestar especial atención a las extensiones relacionadas con el uso de la clave de la TSA.

```
[ tsa_crt ]
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = nonRepudiation, digitalSignature
extendedKeyUsage = critical,timeStamping
```

Tras configurar el entorno de la TSA, generar la clave privada, la CSR y el certificado que se asocia a la TSA.

```
$ openssl req -new -newkey rsa:2048 -keyout private/tsa.key.pem \
  -out tsa.csr.pem -nodes
$ openssl ca -in tsa.csr.pem -extensions tsa_crt -out tsa.crt.pem -days 3650
   -notext
$ echo 01 > tsaserial
```

En este punto se tiene lista la estructura de la TSA. Generar una petición de sellado de tiempo.

```
$ openssl ts -query -data <file_to_hash> -cert -sha1 -out tsarequest.tsq
$ openssl ts -query -digest <digest_bytes_hex> -out tsarequest.tsq
$ openssl ts -query -in tsarequest.tsq -text
$ openssl asn1parse -in tsarequest.tsq -inform DER
```

Remitir la petición.

```
$ openssl ts -reply -queryfile tsarequest.tsq -out tsareply.tsr
$ openssl ts -reply -in tsareply.tsr -text
$ openssl asn1parse -in tsareply.tsr -inform DER
```

Separar el token de la respuesta y comprobar que contiene unicamente `ContentInfo` en lugar de `TimeStampResp`.

```
$ openssl ts -reply -in tsareply.tsr -token_out -out tsatoken.der
$ openssl ts -reply -token_in -in tsatoken.der -text
$ openssl asn1parse -in tsatoken.der -inform DER
```

Queda ya únicamente realizar el procedimiento por el que se comprueba que el sello de tiempo es correcto. Dicha verificación se puede hacer a partir de la respuesta o token obtenido.

```
$ openssl ts -verify -data <file> -in tsareply.tsr -CAfile ca.crt.pem
$ openssl ts -verify -data <file> -token_in -in tsatoken.der \
  -CAfile ca.crt.pem
```

4.7.2. TSA comercial

Múltiples CA comerciales ofrecen también sus servicios como TSA. Algunas de ellas son:

```
http://tsa.safecreative.org
http://tsa.starfieldtech.com
http://timestamp.comodoca.com/authenticode
http://timestamp.globalsign.com/scripts/timestamp.dll
```

No se garantiza que todas las TSA incluidas en el listado anterior sean operativas de la forma que se describe a continuación. En el momento de realización de esta práctica la TSA de Safe Creative opera correctamente, ofreciendo un servicio gratuito de 5 sellos de tiempo por dirección IP y día. Además en su página web ofrece la posibilidad de visualizar los últimos sellos de tiempo que se han realizado e incluso dispone de una herramienta de búsqueda.

Aunque a continuación se trabaja con la TSA de Safe Creative, la metodología es generíca. Para solicitar un sellado de tiempo, OpenSSL ofrece un script de Perl denominado `tsget` que facilita la generación y envío de la petición.

```
$ wget http://tsa.safecreative.org/certificate -O SafeCreative_TSA.cer
$ openssl ts -query -data <file> -cert -sha1 -out SafeCreative.tsq
$ /usr/lib/ssl/misc/tsget -v -h http://tsa.safecreative.org SafeCreative.tsq
$ openssl ts -verify -data <file> -in SafeCreative.tsr \
  -CAfile SafeCreative_TSA.cer
```

Realizar la captura de la petición de sellado de tiempo y analizar su contenido.

4.8. Intercambio seguro de información

A lo largo de esta práctica se ha trabajado con las herramientas proporcionadas por OpenSSL para desplegar las entidades que conforman una PKI. Estos procedimientos extienden los procedimientos criptográficos vistos en la Práctica 1.

Cuestión IX

1) Empleando los conocimientos adquiridos en esta práctica se pide definir, implementar y validar un procedimiento mediante el cual se realice el envío seguro de información entre dos nodos, garantizando la confidencialidad, la integridad, la autenticación y el no repudio.

2) Plantear la manera de desplegar una PKI basada en una jerarquía de CA multinivel, con mínimo una CA raíz y una CA intermedia. Esta última será la que emitirá los certificados de entidad final.

5. Evaluación

Será necesario entregar una pequeña memoria descriptiva de los tareas realizadas durante la práctica en la que se muestre la resolución de las cuestiones planteadas.

Haciendo uso del procedimiento planteado en la cuestión 4.8, remitir los resultados al profesor responsable por correo electrónico. Para evitar posibles errores en el proceso y que sea imposible evaluar el resto de las cuestiones, se sugiere se remitan primeramente los resultados de forma no segura.

Se podrá realizar un examen tipo test tras la práctica o bien incluir alguna pregunta en el examen final con el objetivo que evaluar la comprensión de los conceptos trabajados.

Nota: Los asuntos de los correos electrónicos relacionados con esta práctica deberán contener la cadena `[SRC15_PR2]` *seguida de la descripción breve del mensaje.*

A Fichero `openssl.cnf`

```
#
# Sample OpenSSL configuration
#

####################################################################
# Modify according to your needs

CA_DIR = .
RANDFILE = $ENV::CA_DIR/.rnd

KEY_SIZE = 2048
KEY_COUNTRY = ES
KEY_PROVINCE = Cantabria
KEY_CITY = Santander
KEY_ORG = Universidad de Cantabria
KEY_ORGUNIT = Seguridad en redes de comunicaciones
KEY_EMAIL =

####################################################################

openssl_conf = openssl_init

[ openssl_init ]

oid_section = new_oids
engines = engine_section

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7


[ engine_section ]

[ ca ]

default_ca = CA_default

[ CA_default ]

# Where everything is kept
```

```
dir = $ENV::CA_DIR
# Where the issued certs are kept
certs = $dir/certs
# Database index file
database = $dir/index.txt
# Default place for new certs.
new_certs_dir = $certs
# The CA certificate
certificate = $dir/ca.crt.pem
# The current serial number
serial = $dir/serial

# Where the issued crl are kept
crl_dir = $dir/crl
# The current crl number
crlnumber = $dir/crlnumber
# The current CRL
crl = $dir/crl.pem

# The private key
private_key = $dir/private/ca.key.pem
# Private random number file
RANDFILE = $dir/private/.rand

# The extentions to add to the cert
x509_extensions = usr_cert
# How long to certify for
default_days = 365

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
crl_extensions = crl_ext
# How long before next CRL
default_crl_days = 30

# IMPORTANT: The next must no longer be md5
# Use public key default
default_md = default
# Keep passed DN ordering
preserve = no

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy = policy_match

# For the CA policy
[ policy_match ]

countryName = match
stateOrProvinceName = match
organizationName = match
```

```
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]

countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional


[ req ]

default_bits = $ENV::KEY_SIZE
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

[ req_distinguished_name ]

countryName = Country Name (2 letter code)
countryName_default = $ENV::KEY_COUNTRY
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = $ENV::KEY_PROVINCE

localityName = Locality Name (eg, city)
localityName_default = $ENV::KEY_CITY

0.organizationName = Organization Name (eg, company)
0.organizationName_default = $ENV::KEY_ORG

organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = $ENV::KEY_ORGUNIT
```

```
157
158 commonName = Common Name (eg, your name or your server\'s hostname)
159 commonName_max = 64
160
161 emailAddress = Email Address
162 emailAddress_default = $ENV::KEY_EMAIL
163 emailAddress_max = 40
164
165 [ req_attributes ]
166
167 challengePassword = A challenge password
168 challengePassword_min = 4
169 challengePassword_max = 20
170 unstructuredName = An optional company name
171
172 # These extensions are added when 'ca' signs a request.
173 [ usr_cert ]
174
175
176
177
178
179 [ server_cert ]
180
181
182
183
184
185 # Extensions to add to a certificate request
186 [ v3_req ]
187
188 basicConstraints = CA:FALSE
189 keyUsage = nonRepudiation,digitalSignature,keyEncipherment
190
191 # Extensions for a typical CA
192 [ v3_ca ]
193
194 subjectKeyIdentifier = hash
195 authorityKeyIdentifier = keyid:always,issuer:always
196
197 # This is what PKIX recommends but some broken software chokes on critical
198 # extensions.
199 #basicConstraints = critical,CA:true
200 # So we do this instead.
201 basicConstraints = CA:true
202
203 # Key usage: this is typical for a CA certificate. However since it will
204 # prevent it being used as an test self-signed certificate it is best
205 # left out by default.
206 keyUsage = cRLSign, keyCertSign
207
208 # CRL extensions
209 [ crl_ext ]
```

```
210
211 authorityKeyIdentifier = keyid:always,issuer:always
212
213 [ tsa ]
214
215 # the default TSA section
216 default_tsa = tsa_config1
217
218 # These are used by the TSA reply generation only.
219 [ tsa_config1 ]
220
221 # TSA root directory
222 dir = $ENV::CA_DIR
223 # The current serial number (mandatory)
224 serial = $dir/tsaserial
225 # OpenSSL engine to use for signing
226 crypto_device = builtin
227 # The TSA signing certificate
228 signer_cert = $dir/tsa.crt.pem
229                       # (optional)
230 # Certificate chain to include in reply
231 certs = $dir/cacert.pem
232                       # (optional)
233 # The TSA private key (optional)
234 signer_key = $dir/private/tsakey.pem
235
236 # Policy if request did not specify it
237 default_policy = tsa_policy1
238
239 # acceptable policies (optional)
240 #other_policies = tsa_policy2, tsa_policy3
241
242 # Acceptable message digests (mandatory)
243 digests = md5, sha1
244                       # (optional)
245 accuracy = secs:1, millisecs:500, microsecs:100
246 # number of digits after dot. (optional)
247 clock_precision_digits = 0
248 # Is ordering defined for timestamps?
249 ordering = yes
250                       # (optional, default: no)
251
252 # Must the TSA name be included in the reply?
253 tsa_name = yes
254                       # (optional, default: no)
255
256 # Must the ESS cert id chain be included?
257 ess_cert_id_chain = no
258                       # (optional, default: no)
```