Anexo 1

Creación de certificados de clave pública con OpenSSL

1 OPENSSL 2
2 CERTIFICADOS DE CLAVE PÚBLICA 3
3 FORMATOS DE CLAVES Y CERTIFICADOS 3
4 CREACIÓN DE UNA CLAVE PRIVADA 4
5 GENERACIÓN DE UNA SOLICITUD DE FIRMA DE CERTIFICADO 5
6 EMISIÓN DEL CERTIFICADO 6
6.1 Certificado firmado por una CA no reconocida 7
6.2 Certificados autofirmados 7
7 IMPORTAR CLAVES Y CERTIFICADOS OPENSSL A UN ALMACÉN DE CLAVES DE JAVA 8
7.1 Importar un certificado 8
7.2 Importar una clave privada 8

1 OpenSSL

OpenSSL es un proyecto de software libre que consiste en un paquete de herramientas de administración y bibliotecas relacionadas con la criptografía que se puede usar para:

- Creación claves privadas, claves públicas y parámetros de clave.
- Realización de operaciones criptográficas de clave pública.
- Creación de certificados X.509, solicitudes de firma de certificado (*CSR, Certificate Signing Request*) y listas de revocación de certificados (*CRL, Certificate Revocation List*).
- Cálculo de resúmenes de mensajes (*Message Digests*).
- Cifrado y descifrado con diferentes tipos de *Ciphers*.
- Testeo de clientes y servidores SSL/TLS.
- Manejo de correo S/MIME firmado o cifrado.
- Generación y verificación de solicitudes de marcas temporales (*Time Stamps*).

OpenSSL se incluye por defecto en Linux/Unix y macOS. En este enlace se explica cómo instalarlo en Windows.

Es importante conocer qué versión de OpenSSL vamos a usar antes de realizar cualquier tarea relacionada con la creación de certificados. La versión de OpenSSL que estemos utilizando determina qué algoritmos criptográficos están disponibles, así como qué protocolos están soportados. Por ejemplo, la versión 1.0.1 fue la primera que soportaba TLS 1.1 y TLS 1.2. Se puede saber que versión de OpenSSL está disponible en el sistema ejecutando el comando siguiente:

```
# openssl version -a
```

El parámetro -a se utiliza para mostrar toda la información de versión. A continuación, se muestra la salida de este comando, resaltando en color más oscuro la información siguiente:

- El número de versión y su fecha de lanzamiento.
- Las opciones que fueron incluidas en la librería.
- El directorio por defecto para guardar certificados, claves privadas y ficheros de configuración.

```
OpenSSL 1.1.1d  10 Sep 2019
built on: Sat Oct 12 19:56:43 2019 UTC
platform: debian-amd64
options: bn(64,64) rc4(16x,int) des(int) blowfish(ptr)
compiler: gcc  fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack
-g -O2 -fdebug-prefix-map=/build/openssl-YwazYa/openssl-1.1.1d=.
-fstack-protector-strong -Wformat -Werror=format-security -DOPENSSL_USE_NODELETE
-DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2
-DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM
-DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM
-DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG
-Wdate-time -D_FORTIFY_SOURCE=2
OPENSSLDIR: "/usr/lib/ssl"
ENGINESDIR: "/usr/lib/x86_64-linux-gnu/engines-1.1"
Seeding source: os-specific
```

Si se invoca sin parámetros se ejecuta en modo interactivo mostrando su propia línea de comando, en la que se pueden ejecutar cualquiera de los comandos disponibles:

```
# openssl
Openssl> version -a
```

2 Certificados de clave pública

Un certificado de clave pública es un tipo de documento digital que vincula los datos de su titular con una clave pública. Para que ese vínculo quede legalmente establecido, el certificado tiene que estar firmado digitalmente por una Autoridad de Certificación o CA (*Certification Authority*).

En criptografía, una CA es una entidad de confianza, responsable de emitir y revocar certificados digitales usando criptografía de clave pública. Jurídicamente es un caso particular de Prestador de Servicios de Certificación. La autoridad de la CA garantiza que cualquier uso de la clave privada se puede vincular de forma inequívoca al titular del certificado asociado.

Los certificados de clave pública contienen una serie de campos, siendo los más habituales:

- Serial Number: número de serie que identifica de forma única el certificado de la CA. Se usa para llevar a cabo su revocación.
- Subject: Indica la entidad que se certifica (p. ej. una máquina, una persona o una organización)
- Issuer: La entidad que verificó la información y firmó el certificado (la Autoridad de certificación)
- Not Before: El momento a partir del cual el certificado es válido.
- Not After: El momento a partir del cual el certificado no es válido.
- Key Usage: Los usos criptográficos válidos para los que se puede usar el certificado (p. ej. firma de certificados, cifrado de claves, cifrado de datos, o validación de firmas digitales).
- Extended Key usage: Aplicaciones en las cuales puede usar el certificado (p. ej. autenticación de servidores TLS, protección de correo electrónico o firma de código)
- Public Key: La clave pública firmada en el certificado.
- Signature Algorithm: Algoritmo usado para firmar el certificado de clave pública.
- Signature: La firma del cuerpo del certificado por la Autoridad de certificación.

La generación de un certificado conlleva la realización de una serie de pasos:

- Creación de una solicitud de firma, que contiene la clave pública y los datos de identificación del titular.
- Envío de la solicitud a una CA que, por sí misma o mediante la intervención de una Autoridad de Registro, verifica la identidad del titular que figura en la solicitud.
- Una vez verificada la identidad del titular, la CA, con su clave privada, firma la solicitud para completar la emisión del certificado.
- Finalmente, entrega del certificado al titular.

3 Formatos de claves y certificados

X.509 es un conjunto de estándares y protocolos utilizados en la infraestructura de clave pública o PKI (*Public Key Infrastructure*). Entre estos estándares y protocolos, se definen el formato que deben tener los certificados, las claves públicas y las claves privadas. Estos certificados y las claves, en función del formato elegido, se pueden almacenar en un mismo fichero o de forma individual.

Para almacenar certificados y claves se han definido muchos formatos estándar que rivalizan entre sí, de los cuales, los más comunes son:

- DER: almacena claves y certificados en formato binario usando la codificación DER ASN.1. La extensión de los ficheros suele ser *.der*.
- PEM: almacena claves y certificados codificados en Base64 en archivos ASCII. La codificación en Base64 se delimita por las líneas `-----BEGIN RSA PRIVATE KEY-----` y `-----END RSA PRIVATE KEY-----`

para las claves, con ligeras variaciones en función de los algoritmos utilizados, y por las líneas `-----BEGIN CERTIFICATE-----` y `-----END CERTIFICATE-----` para los certificados. La extensión utilizada suele ser *.pem*.

- PKCS #8: es una especificación que define la estructura de claves privadas, que aún han de ser codificadas usando el formato DER o el formato PEM. La extensión estándar es *.p8*.
- PKCS #12: es el formato más extendido para almacenar claves y certificados en formato encriptado. Se puede usar incluso para almacenar contraseñas arbitrarias. Se utiliza por defecto en las versiones más recientes de Java para crear almacenes de claves. La extensión estándar es *.p12* o *.pfx* (PFX es un predecesor de PKCS12).

4 Creación de una clave privada

Para crear una clave privada con OpenSSL se ejecutará uno de los comandos siguientes, dependiendo del algoritmo que se desee usar:

```
# openssl genrsa …

# openssl gendsa …

# openssl genpkey …
```

Por ejemplo, el comando siguiente crea una clave privada RSA de 3072 bits de longitud y la almacena sin cifrar en el fichero `rsa_priv.pem` con el formato por defecto usado por OpenSSL:

```
# openssl genrsa -out rsa_priv.pem 3072
```

Para usar la clave en Java hay que almacenarla usando la especificación PKCS8 codificada en formato DER. El comando siguiente realiza la conversión de la contraseña generada anteriormente y la almacena en el fichero `rsa_priv.p8`:

```
# openssl pkcs8 -topk8 -inform PEM -in rsa_priv.pem -outform DER -out rsa_priv.p8 -nocrypt
```

El fragmento de código Java que se muestra a continuación crea un objeto `PrivateKey` a partir del contenido del fichero `rsa_priv.p8`:

```
PrivateKey privada;
try (DataInputStream in = new DataInputStream(new DataInputStream("rsa_priv.p8"))) {
    byte[] key = new byte[in.available()];
    in.read(key);
    PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(key);
    KeyFactory kf = KeyFactory.getInstance("rsa");
    privada = kf.generatePrivate(spec);
}
```

Para crear una clave privada cifrada, se ha de especificar en el momento de su creación un algoritmo de cifrado y una contraseña o *pass phrase*. El algoritmo de cifrado se especifica mediante uno de los parámetros `-aes128`, `-aes192`, `-aes256`, `-aria128`, `-aria192`, `-aria256`, `-camellia128`, `-camellia192`, `-camellia256`, `-des`, `-des3` o `-idea`. La *pass phrase* se especifica con el parámetro `-passout`. Si se omite este parámetro, se le pedirá al usuario que la introduzca a través de la consola. Por ejemplo:

```
# openssl genrsa -out rsa_priv.pem -aes128 3072
Generating RSA private key, 3072 bit long modulus (2 primes)
.....................++++
.....................................................................................++++
```

```
e is 65537 (0x010001)
Enter pass phrase for miclave:
Verifying - Enter pass phrase for miclave:
```

Con el uso del parámetro `-passout`, la *pass phrase* se puede aportar de tres formas diferentes:

`-passout stdin`	se lee de la entrada estándar
`-passout pass:`*`contraseña`*	se usa *`contraseña`* como *pass phrase*
`-passout file:`*`fichero`*	se lee del fichero de texto plano *`fichero`*

Para ver el material de clave almacenada con formato PEM se ejecuta el comando siguiente:

```
# openssl rsa -text -in rsa_priv.pem -noout
```

El parámetro `-noout` hace que se omita en la salida la codificación Base64 de la clave.

A partir del archivo que contiene la clave privada se puede obtener la clave pública asociada ejecutando el comando siguiente:

```
# openssl rsa -in rsa_priv.pem -pubout -out rsa_pub.pem
```

Para usar la clave pública en Java hay que convertirla a formato DER:

```
# openssl rsa -pubin -in rsa_pub.pem -inform PEM -pubout -out rsa_pub.der -outform DER
```

De forma alternativa, se puede obtener directamente a partir de la clave privada, una clave pública en formato DER para que pueda ser usada en Java:

```
# openssl rsa -in rsa_priv.pem -pubout -outform DER -out rsa_pub.der
```

El fragmento de código Java que se muestra a continuación crea un objeto `PublicKey` a partir del contenido del fichero que almacena la clave pública en formato DER:

```
PublicKey publica;
try (DataInputStream in = new DataInputStream(new FileInputStream("rsa_pub.der"))) {
    byte[] key = new byte[in.available()];
    in.read(key);
    X509EncodedKeySpec spec = new X509EncodedKeySpec(key);
    KeyFactory kf = KeyFactory.getInstance("rsa");
    publica = kf.generatePublic(spec);
}
```

5 Generación de una solicitud de firma de certificado

Para obtener un certificado SSL es necesario crear una solicitud de firma de certificado (CSR) que posteriormente se enviará a una Autoridad de Certificación (CA). Se necesitará una clave privada de la cual extraer la clave pública que se va a incorporar a la solicitud de firma. Podemos utilizar una clave privada existente o podemos generar una clave privada nueva.

Para crear un CSR a partir de una clave privada almacenada en el fichero `rsa_priv.pem`, se ejecuta el comando siguiente:

```
# openssl req -new -key rsa_priv.pem -out cert.csr
```

Este comando solicitará a través de la entrada estándar los datos del titular que se incorporarán a la solicitud junto a la clave pública asociada a la clave privada especificada (no es necesario extraerla previamente). Estos datos son:

`Country Name`	Un código de dos caracteres del país donde se localiza legalmente la organización a la que pertenece el titular.
`State or Province`	En función del país, el estado o provincia donde se localiza legalmente la organización a la que pertenece el titular.
`Locality Name`	La ciudad o localidad donde se localiza legalmente la organización a la que pertenece el titular.
`Organization Name`	El nombre legalmente registrado de la organización a la que pertenece el titular.
`Organizational Unit Name`	Departamento dentro de la organización (se puede dejar en blanco).
`Common Name`	Nombre del titular. Para certificados de servidor, el FQDN (***F**ully-**Q**ualified **D**omain **N**ame*). Ejemplo, www.midominio.org.
`Email Address`	Dirección de correo electrónico (se puede dejar en blanco).
`A challenge password`	Se puede dejar en blanco.
`An optional company name`	Se puede dejar en blanco.

A los datos que se dejen en blanco pulsando directamente la tecla INTRO, se les asignará un valor por defecto que se define en el fichero de configuración de OpenSSL.

Otras formas de crear una solicitud de firma son:

- Especificando los datos del titular en la línea de comando mediante el parámetro `-subj`:

```
# openssl req -new -key rsa_priv.pem -out cert.csr -subj "/C=ES/ST=ASTURIAS/L=SOTRONDIO/O=DAM/CN=PSP"
```

- Generando la clave privada y la CSR en un solo paso:

```
# openssl req -new -newkey rsa:3072 -nodes -keyout rsa_priv.pem -out cert.csr
```

En este caso se crea una nueva clave privada RSA de 3072 bits de longitud, sin cifrar (-nodes), y se almacena en el fichero `rsa_priv.pem`. A continuación, se genera la solicitud de firma y se almacena en el fichero `cert.csr`. Aquí también existe la posibilidad de especificar los datos del titular en la línea de comando mediante el parámetro -subj.

6 Emisión del certificado

Si queremos un certificado firmado por una Autoridad de Certificación de confianza, tendremos que enviarle la solicitud de firma en formato PEM. La CA no emitirá el certificado hasta haber comprobado la identidad del titular de la solicitud.

La emisión de un certificado por parte de una CA de confianza conlleva en muchos casos el correspondiente desembolso económico, que varía en función del tipo de certificado que se solicite. Sin embargo, existen situaciones en las que no se requiere que la firma proporcione el nivel de confianza que se necesita en ámbitos como el comercio electrónico, la banca electrónica, la realización de trámites con administraciones públicas, etc. Es el caso, por ejemplo, del uso de certificados para la realización de pruebas durante el desarrollo de software. A continuación, se exponen dos formas de emitir certificados que no estarán firmados por una CA de confianza para su uso en este tipo de situaciones.

6.1 Certificado firmado por una CA no reconocida

El primer paso para emitir un certificado de este tipo a partir de la CSR correspondiente, es crear una CA:

```
# openssl req -newkey rsa:3072 -x509 -keyout cakey -out ca.crt -days 3650 -nodes -subj
"/C=ES/ST=ASTURIAS/L=SOTRONDIO/O=DAM/CN=DAMCA"
```

Este comando crea el archivo `ca.crt` que contiene el certificado de la nueva CA y el archivo `cakey` que contiene su clave privada, con la que se firmarán las CSRs. Por ejemplo:

```
# openssl x509 -req -in cert.csr -days 3650 -CA ca.crt -CAkey cakey -set_serial 01 -out
micertificado.crt
```

6.2 Certificados autofirmados

Un certificado autofirmado es aquel que se firma con la misma clave privada con la que se crea la solicitud de firma. Por ejemplo:

```
# openssl x509 -req -in cert.csr -days 3650 -signkey rsa_priv.pem -out micertificado.crt
```

El comando siguiente muestra los datos del cetificado autofirmado. Se puede observar que los campos `Issuer` y `Subject` coinciden por tratarse de un certificado autofirmado:

```
# openssl x509 -text -in micertificado.crt -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            7d:03:3e:40:c4:41:71:2f:64:b4:fc:f7:82:16:ae:af:2e:3b:eb:67
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = ES, ST = ASTURIAS, L = SOTRONDIO, O = DAM, CN = PSP
        Validity
            Not Before: Mar  6 13:04:39 2022 GMT
            Not After : Mar  3 13:04:39 2032 GMT
        Subject: C = ES, ST = ASTURIAS, L = SOTRONDIO, O = DAM, CN = PSP
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (3072 bit)
                Modulus:
                    00:e5:d0:96:5f:08:4f:27:d1:7f:ed:ac:e8:3b:68:
                    99:e5:23:40:f4:f7:8a:9d:09:e3:fd:91:3c:9d:9f:
                    01:fb:e1:b5:8e:d4:82:0f:d0:a5:84:1a:07:f9:48:

                                        ⋮

                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         3c:12:88:0a:58:ce:03:7f:dc:35:ea:38:45:12:d3:aa:cb:b7:
         9a:62:41:85:91:72:64:29:2a:42:26:4c:63:26:21:e9:0a:2e:
         8c:18:ba:51:5a:3a:32:08:13:3e:b5:79:63:64:09:71:51:68:

                                    ⋮
```

7 Importar claves y certificados OpenSSL a un almacén de claves de Java

Las claves privadas y certificados generados con OpenSSL se pueden importar a un almacén de claves de Java usando la herramienta keytool. En este apartado se usarán como ejemplo claves privadas y certificados generados en los apartados anteriores.

7.1 Importar un certificado

Los certificados se importan directamente usando el comando `keytool -importcert`. Para importar, por ejemplo, el certificado almacenado en el fichero `micertificado.crt` al almacén de claves `keystore.p12`, se ejecuta el comando siguiente:

```
# keytool -importcert -keystore keystore.p12 -alias micertificado -file micertificado.crt
```

7.2 Importar una clave privada

En el caso de las claves privadas, no es posible importarlas directamente a un almacén de claves. En su lugar, hay que almacenar la clave y un certificado asociado a la misma en un fichero PKCS#12 para importarlo al almacén de claves. Por ejemplo, dados el fichero `rsa_priv.pem` que almacenta una clave privada y el fichero `micertificado.crt` que almacena un certificado asociado a dicha clave, se ejecuta el comando OpenSSL siguiente:

```
# openssl pkcs12 -export -in micertificado.crt -inkey rsa_priv.pem -out nuevo.p12 -name miclave
Enter Export Password:
Verifying - Enter Export Password:
```

Una vez introducida y verificada la contraseña de exportación (`Export Password`) se crea el fichero `nuevo.p12`, que para keytool será otro almacén de claves más. El valor especificado en el parámetro `-name` servirá como alias para la herramienta keytool. Finalmente, para importar `nuevo.p12` al almacén de claves `keystore.p12` habría que ejecutar el comando siguiente:

```
# keytool -importkeystore -destkeystore keystore.p12 -srckeystore nuevo.p12 -srcstoretype PKCS12
Importando el almacén de claves de nuevo.p12 a keystore.p12...
Introduzca la contraseña de almacén de claves de destino:
Introduzca la contraseña de almacén de claves de origen:
La entrada del alias pedro2 se ha importado correctamente.
Comando de importación completado: 1 entradas importadas correctamente, 0 entradas incorrectas o
canceladas
```

Para lleva a cabo la importación, se pide la contraseña del almacén de claves de destino y la contraseña del almacén de claves de origen (la que se especificó como contraseña de exportación al crear `nuevo.p12` con OpenSSL).