Extraer PEM con OpenSSL

Extraer el certificado y la clave privada con OpenSSL

Anonymous · 02/09/2026

Extraer el certificado y la clave privada con OpenSSL

Para autenticar tus peticiones a la API necesitas dos valores en formato PEM: el certificado (con su cadena de certificación) y la clave privada. Ambos salen del archivo `.p12` o `.pfx` de tu firma electrónica.

Puedes obtenerlos con la herramienta externa tools.libredte.cl o, si prefieres que tu firma no salga de tu computador, extraerlos tú con **OpenSSL**. Este tutorial cubre la segunda opción: todos los comandos se ejecutan localmente y tu archivo no se sube a ningún servicio.

Antes de empezar

Necesitas OpenSSL instalado y el archivo de tu firma con su contraseña.

```
openssl version
```

- **macOS**: viene incluido; si usas Homebrew, `brew install openssl`.
- **Linux (Debian/Ubuntu)**: `sudo apt install openssl`.
- **Windows**: usa Git Bash, WSL o los binarios de slproweb.com/products/Win32OpenSSL.html.

En los ejemplos el archivo de la firma se llama `firma.p12`. Reemplaza ese nombre por el tuyo. OpenSSL te pedirá la contraseña de la firma en cada comando.

Paso 1: extraer el certificado con su cadena

Este es el valor que enviarás como `cert-data`. La opción `-nokeys` excluye la clave privada y, al no usar `-clcerts`, la salida incluye tanto tu certificado como los de la autoridad certificadora (la cadena).

```
openssl pkcs12 -legacy -in firma.p12 -nokeys \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > cert-data.txt
```

El archivo `cert-data.txt` queda con el valor listo para copiar, en una sola línea. Debe contener dos o

más bloques `BEGIN CERTIFICATE`, en el orden correcto: primero tu certificado y después la cadena. Puedes confirmarlo con:

```
grep -o "BEGIN CERTIFICATE" cert-data.txt | wc -l
```

> **Importante:** envía el valor completo con todos sus bloques. Si mandas solo el primer certificado, la autenticación puede ser rechazada por falta de la cadena de certificación.

Paso 2: extraer la clave privada

Este es el valor que enviarás como `pkey-data`. La opción `-nocerts` excluye los certificados y `-nodes` deja la clave sin cifrar, que es como debe viajar en la petición.

```
openssl pkcs12 -legacy -in firma.p12 -nocerts -nodes \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > pkey-data.txt
```

`pkey-data.txt` debe empezar con `-----BEGIN PRIVATE KEY-----` o con `-----BEGIN RSA PRIVATE KEY-----`; la API acepta ambos encabezados.

Si la clave quedó cifrada

Si el archivo empieza con `-----BEGIN ENCRYPTED PRIVATE KEY-----`, la API la rechazará. Revisa primero que no hayas omitido la opción `-nodes`. Si el problema persiste, hay que descifrarla en un archivo intermedio antes de convertirla a una línea:

```
openssl pkcs12 -legacy -in firma.p12 -nocerts -nodes \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' > pkey_temp.pem
openssl pkcs8 -topk8 -nocrypt -in pkey_temp.pem -out pkey_plano.pem
LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' pkey_plano.pem > pkey-data.txt
rm pkey_temp.pem pkey_plano.pem
```

Paso 3: verificar lo que extrajiste

Como los archivos están en una sola línea, `openssl` no puede leerlos directamente: hay que devolver los saltos con un `awk` antes de pasárselos.

Confirma que el certificado es el que esperas y que está vigente:

```
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' cert-data.txt \
  | openssl x509 -noout -subject -issuer -dates
```

Verás el titular, la autoridad certificadora y las fechas de validez. Para comprobar que la clave privada corresponde a ese certificado, ambos módulos deben coincidir:

```
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' cert-data.txt | openssl x509 -noout -modulus
| openssl md5
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' pkey-data.txt | openssl rsa -noout -modulus
| openssl md5
```

Si los dos comandos devuelven el mismo hash, el par es correcto.

Paso 4: enviar los valores a la API

Los dos valores viajan en el bloque `auth.cert` del cuerpo JSON, en cada petición:

```
{
  "auth": {
    "cert": {
      "cert-data": "-----BEGIN CERTIFICATE-----\n...",
      "pkey-data": "-----BEGIN PRIVATE KEY-----\n..."
    }
  }
}
```

Abre `cert-data.txt` y `pkey-data.txt`, y copia el contenido completo de cada uno en el campo que corresponde. Son valores de una sola línea, así que no necesitas escaparlos ni reformatearlos: sirven igual escribiendo el JSON a mano (Postman, Swagger, curl) que armando la petición por código.

Qué hace cada parte del comando

Los comandos de los pasos 1 y 2 encadenan tres piezas. No necesitas entenderlas para seguir el tutorial, pero saber qué hace cada una te ayudará si algo falla.

La opción `-legacy`

Las firmas electrónicas chilenas vienen cifradas con algoritmos antiguos (RC2 de 40 bits, SHA-1) que OpenSSL 3.x desactivó por defecto, así que sin esa opción los comandos fallan con un error como este:

```
Error outputting keys and certificates
...:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:
crypto/evp/evp_fetch.c:375:Global default library context,
Algorithm (RC2-40-CBC : 0), Properties ()
```

Si usas OpenSSL 1.1.x, la opción `-legacy` no existe y tampoco hace falta: quítala de los comandos y funcionarán igual.

El filtro de las líneas `Bag Attributes`

OpenSSL no entrega el PEM limpio: antes de cada bloque intercala metadatos como `Bag Attributes`, `localKeyID`, `subject=` e `issuer=`. Se ven así:

```
Bag Attributes
    localKeyID: 87 F1 CE 34 18 B5 C4 41 A9 9C 20 9E FC 39 D6 08 67 FC B8 0A
subject=CN=Titular Test
issuer=CN=CA Test
-----BEGIN CERTIFICATE-----
```

Esas líneas **deben quedar fuera** de lo que envíes a la API. No es un detalle estético: la API valida que el valor empiece exactamente con `-----BEGIN CERTIFICATE-----` y, si no, rechaza la petición con el error *"El certificado no tiene el formato PEM correcto"*.

`openssl pkcs12` no tiene ninguna opción para omitir esos metadatos, así que los comandos pasan la salida por `sed`, que conserva únicamente lo que va desde `-----BEGIN` hasta `-----END`.

El prefijo `LC_ALL=C` que verás en esos comandos hace que `sed` trate la entrada como bytes sueltos. Es necesario porque los metadatos incluyen el nombre del titular y, si lleva acentos o ñ en una codificación distinta de UTF-8, el `sed` de macOS y de los BSD aborta con `RE error: illegal byte sequence`. En Linux con GNU `sed` el prefijo es inofensivo, así que puedes usar el mismo comando en cualquier sistema.

Si trabajas en Windows sin `sed` (fuera de Git Bash o WSL), ejecuta el comando sin la tubería y luego borra a mano, en un editor de texto, todo lo que esté antes de la primera línea `-----BEGIN`.

La conversión a una línea

La API no recibe el PEM multilínea tal cual: espera el valor en **una sola línea**, con los saltos representados como los dos caracteres `\` y `n`. Así se ve el resultado:

```
-----BEGIN CERTIFICATE-----\nMIIC7zCCAdegAwIBAgIUC3UApaj1...\n-----END CERTIFICATE----
-\n
```

Por eso los comandos terminan con un `awk` que hace esa conversión: cada uno genera un único archivo, ya listo para pegar sin retoques, tanto si armas la petición por código como si escribes el JSON a mano.

Si OpenSSL falla

- **`Mac verify error: invalid password?`** — la contraseña no corresponde al archivo. Es la causa más frecuente; revísala antes de buscar otra explicación.
- **`unsupported`** o **`RC2-40-CBC`** — falta la opción `-legacy` en el comando.
- **`sed: RE error: illegal byte sequence`** — falta el prefijo `LC_ALL=C` antes de `sed`. Ocurre en macOS y BSD cuando el nombre del titular en los metadatos del archivo trae acentos o ñ en una codificación distinta de UTF-8.
- **`No such file or directory`** — revisa la ruta del archivo. Si el nombre tiene espacios, enciérralo en comillas.

`could not load the shared library` al usar `-legacy`

El módulo del proveedor `legacy` no está disponible. Comprueba si tu instalación lo tiene:

```
openssl list -providers -provider legacy
```

Si responde `status: active`, el proveedor está bien y el problema es otro. Si falla, el módulo no está instalado o el `openssl` de tu `PATH` no es el mismo al que pertenecen los módulos instalados, algo habitual cuando hay más de un OpenSSL en el sistema. Revisa dónde los busca con:

```
openssl version -m
```

Como alternativa, en vez de `-legacy` puedes cargar los proveedores de forma explícita:

```
openssl pkcs12 -provider legacy -provider default -in firma.p12 -nokeys \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > cert-data.txt
```

Tienen que ir los dos: si indicas solo `-provider legacy`, OpenSSL deja de cargar el proveedor por defecto y falla con `Error verifying PKCS12 MAC; no PKCS12KDF support`.

Cuida tu clave privada

- Es una credencial secreta: quien la tenga puede firmar en tu nombre.
- No la subas a repositorios ni la pegues en canales de chat o tickets.
- Guárdala en variables de entorno o en un gestor de secretos, nunca en el código fuente.
- El archivo `pkey-data.txt` queda **sin cifrar** en tu disco. Bórralo cuando ya no lo necesites, o restringe sus permisos.

Last updated on 02/09/2026