



Manual de Integración

Izenpe S.A. 2025

Este documento es propiedad de Izenpe, s.a. y su contenido es confidencial. Este documento no puede ser reproducido, en su totalidad o parcialmente, ni mostrado a otros, ni utilizado para otros propósitos que los que han originado su entrega, sin el previo permiso escrito de Izenpe, S.A. En el caso de ser entregado en virtud de un contrato, su utilización estará limitada a lo expresamente autorizado en dicho contrato. Izenpe, s.a. no podrá ser considerada responsable de eventuales errores u omisiones en la edición del documento.

Histórico de versiones

Versión	Fecha	Resumen de los cambios producidos
1.0	05/04/2018	Primera versión
1.1	09/05/2018	Se incluye capítulo de <i>Integración con el servicio de autenticación SAML 2.0</i>
1.2	14/09/2018	Se amplían los formatos de firma en la API de firma de documentos: XMLDSig/XAdES Enveloping y CMS/CADES
1.3	30/10/2018	Se incluye nueva columna en los parámetros de firma para distinguir entre Idazki Desktop < v3.0 e Idazki Desktop >= v3.0
1.4	07/01/2019	Se incluye capítulo de <i>Integración con el servicio de consulta de operaciones y eventos</i> , además de un ejemplo en código Java
1.5	12/03/2019	<p>Se incluye descripción de parámetro <i>login_hint</i> al iniciar flujo de autenticación OAuth 2.0.</p> <p>Se incluye descripción de la optimización llevada a cabo en Giltza v4.1.8.0 para las firmas en serie (contrafirma o countersign) XAdES y CMS/CADES:</p> <ul style="list-style-type: none"> • Se añaden los capítulos <i>Contrafirmas XAdES</i> y <i>Contrafirmas CMS/CADES</i> • Se modifica el capítulo <i>Firmas múltiples</i> • Se modifica el nombre de capítulo <i>XAdES Externally Detached en serie (countersign)</i> por <i>XAdES Enveloped en serie I (countersign)</i> • Se añade el capítulo <i>XAdES Enveloped en serie II (countersign)</i>
1.6	18/03/2020	<p>Se incluye el nuevo flujo de autenticación mediante Izenpe Mobile, dentro del capítulo <i>Integración con el servicio de autenticación OAuth 2.0</i>.</p> <p>Se incluyen las descripciones de los nuevos atributos de Izenpe Mobile en los detalles de autenticación devueltos por Giltza, dentro del capítulo <i>Atributos de usuario en OAuth 2.0</i>.</p> <p>Se modifica el punto <i>Kit de integración con el servicio de firma</i> para incluir una llamada opcional a la API de Giltza para obtener la información de un proceso de firma de documentos.</p> <p>Se añade el apartado <i>Obtención de la información de un proceso de firma de documentos (opcional)</i> que describe la manera de explotar la información devuelta por Giltza para saber si se ha iniciado un proceso de actualización a través del mecanismo de firma Idazki Desktop.</p>
1.7	04/11/2020	Se incluye el nuevo flujo de autenticación mediante Cl@ve 2, dentro del capítulo <i>Integración con el servicio de autenticación OAuth 2.0</i> .

1.8	26/11/2020	Se incluyen los nuevos flujos de autenticación mediante Izenpe Mobile, dentro del capítulo <i>Integración con el servicio de autenticación OAuth 2.0</i> .
1.9	11/02/2021	Se detalla el nombre de los flujos de autenticación de Cl@ve 2, dentro del punto 5.1.1.1 <i>Petición</i> .
1.10	06/06/2022	Se añade la descripción del atributo perfil en el JSON de UserInfo, dentro del punto 5.1.4 <i>Atributos de usuario en OAuth 2.0</i> .
1.11	09/02/2023	Se añade el punto 5.1.5 <i>Logout</i> . Se añaden nuevas etiquetas de firma (<i>labels</i>) para habilitar en Idazki Desktop v3.6.0 la comprobación del certificado firmante a través de las TSLs (Trust Service Lists), dentro del punto 7.2.2 <i>Creación del proceso de firma</i> . Se añade descripción de las etiquetas de firma (<i>labels</i>) dinámicas, dentro del punto 7.2.2 <i>Creación del proceso de firma</i> .
1.12	07/03/2023	Se modifica el punto 7.1 <i>Resumen de las operaciones de firma disponibles en Giltza</i> . Se añade el punto 7.3.3.1.4 <i>Firmas XMLDSig/XAdES Enveloping con elemento Manifest</i> con la definición JSON para este tipo de firmas. Se añade el punto 12 <i>Ejemplos de verificación de firmas mediante Zain</i> , en este caso para las firmas con elementos Manifest.
1.13	21/03/2023	Se elimina toda mención a la librería Axis dentro del punto 12 <i>Ejemplos de verificación de firmas mediante Zain</i> .
1.14	09/05/2023	Se elimina capítulo de <i>Integración con el servicio de consulta de operaciones y eventos</i> , además del ejemplo en código Java.
1.15	28/12/2023	Se actualiza el capítulo 1 <i>Introducción</i> y el capítulo 5.1.1 <i>Obtención de la autorización</i> para ofrecer una visión más clara de los diferentes flujos de autenticación y niveles en Giltza. Se añaden las etiquetas correspondientes al uso de Idazki Desktop combinado con certificado en la nube en Giltza Profesional, dentro del apartado 7.2.2 <i>Creación del proceso de firma</i> .
1.16	15/01/2024	Se corrige el apartado 7.3.1 <i>Signer</i> para indicar como campo obligatorio el atributo <i>signature_field.location.page</i> del JSON.
1.17	25/06/2024	Se corrige el flujo de Bak. Se elimina: <i>extended</i> .
1.18	05/03/2025	Se añade la etiqueta de firma <i>serverid</i> dentro de los <i>labels</i> de firma.
1.19	09/04/2025	Se incluye el punto 4.1 <i>URIs de redirección permitidas</i> . Se actualiza el punto 13. <i>Referencias</i> con la URL de RFC6749 "The OAuth 2.0 Authorization Framework".

Contenido

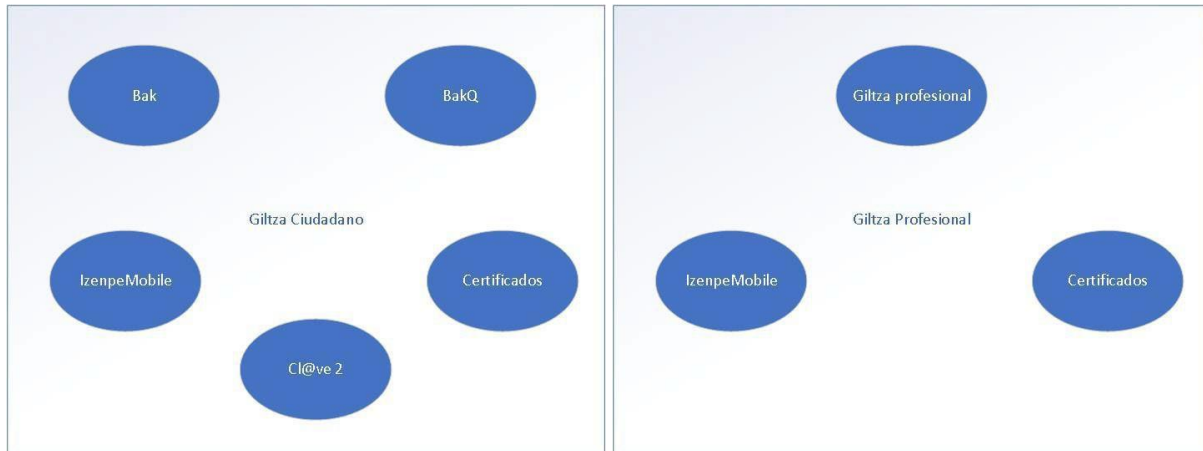
Histórico de versiones.....	2
1. Introducción.....	7
1.1. Objeto del documento.....	7
2. Autorización para utilizar las APIs.....	8
3. Endpoints de Giltza.....	9
4. Registro de una aplicación.....	10
4.1. URIs de redirección permitidas.....	10
5. Integración con el servicio de autenticación OAuth 2.0.....	11
5.1. Kit de integración con el servicio de autenticación.....	11
5.1.1. Obtención de la autorización.....	12
5.1.1.1. Petición.....	12
5.1.1.2. Respuesta.....	18
5.1.2. Obtención del token de acceso.....	18
5.1.2.1. Petición.....	18
5.1.2.2. Respuesta.....	19
5.1.3. Consulta de los datos de usuario.....	20
5.1.3.1. Petición.....	20
5.1.3.2. Respuesta.....	20
5.1.4. Atributos de usuario en OAuth 2.0.....	20
5.1.5. Logout.....	24
5.2. Flujo step-up.....	25
5.3. Diagrama de flujo para autenticación con OAuth2.0.....	26
6. Integración con el servicio de autenticación SAML 2.0.....	27
6.1. Kit de integración con el servicio de autenticación.....	27
6.1.1. Petición de autenticación HTTP.....	27
6.1.1.1. Petición.....	28
6.1.2. Respuesta de autenticación HTTP.....	29
6.1.2.1. Respuesta.....	29
6.1.3. Diagrama de flujo para autenticación con SAML 2.0.....	35
6.1.4. Gestión de errores.....	35
6.1.4.1. Errores sin redirección al proveedor de servicios.....	35
6.1.4.2. Errores con redirección al proveedor de servicios.....	36
6.1.4.3. Ejemplo de respuesta de error.....	36
7. Integración con el servicio de firma.....	37
7.1. Resumen de las operaciones de firma disponibles en Giltza.....	37
7.1.1. Básicas.....	37
7.1.2. Por lotes.....	38
7.1.3. Múltiples.....	38
7.2. Kit de integración con el servicio de firma.....	38
7.2.1. Obtención del token OAuth 2.0.....	41
7.2.1.1. Petición.....	41
7.2.1.2. Respuesta.....	41
7.2.2. Creación del proceso de firma.....	42
7.2.2.1. Petición.....	43
7.2.2.2. Respuesta.....	42
7.2.3. Creación de un recurso.....	43
7.2.3.1. Petición.....	42

7.2.3.2.	Respuesta.....	42
7.2.4.	Ejecución del proceso de firma	42
7.2.4.1.	Petición	42
7.2.4.2.	Respuesta.....	43
7.2.5.	Obtención de la información de un proceso de firma de documentos (opcional)	42
7.2.5.1.	Petición	42
7.2.5.2.	Respuesta.....	42
7.2.6.	Obtención de la información de los documentos firmados	42
7.2.6.1.	Petición	42
7.2.6.2.	Respuesta.....	42
7.2.7.	Obtención del documento firmado	43
7.2.7.1.	Petición	43
7.2.7.2.	Respuesta.....	43
7.2.8.	Eliminación de un recurso	42
7.2.8.1.	Petición	42
7.2.8.2.	Respuesta.....	42
7.2.9.	Eliminación del proceso de firma	42
7.2.9.1.	Petición	42
7.2.9.2.	Respuesta.....	42
7.2.10.	Obtención del resultado de un proceso de firma	43
7.2.10.1.	Petición	43
7.2.10.2.	Respuesta.....	43
7.3.	Configuración de características de la firma.....	42
7.3.1.	Signer	42
7.3.3.1.	Parameters.....	43
7.3.3.1.1.	FirmasPDF/PAdES.....	43
7.3.3.1.2.	Firmas XMLDSig/XAdES Enveloped	44
7.3.3.1.3.	Firmas XMLDSig/XAdES	49
7.3.3.1.4.	Firmas XMLDSig/XAdES	53
7.3.3.1.5.	Firmas XMLDSig/XAdES	59
7.3.3.1.6.	Firmas XMLDSig/XAdES	64
7.3.3.1.7.	Firmas XMLDSig/XAdES	58
7.3.3.1.8.	Contrafirmas XAdES	59
7.3.3.1.9.	Firmas CMS/CADES.....	64
7.3.3.1.10.	Contrafirmas CMS/CADES.....	67
7.4.	Firmas múltiples	70
7.4.1.	PAdES en serie (countersign).....	71
7.4.2.	XAdES Externally Detached en paralelo (cosign).....	71
7.4.3.	XAdES Enveloped en serie I (countersign)	72
7.4.4.	XAdES Enveloped en serie II (countersign)	74
7.5.	Diagrama de flujo para firma.....	77
8.	Federación de identidades	65
8.1.	Configuración	65
9.	Status de las respuestas HTTP	66
9.1.	Códigos de estado de operaciones completadas correctamente	66
9.2.	Códigos de estado con parámetro error en la cabecera HTTP WWW- Authenticate.....	66
9.3.	Códigos de estado con descripción opcional del error en el cuerpo de la respuesta	67
10.	Ejemplos de integración mediante cURL.....	73
10.1.	Servicio de autenticación	73
10.1.1.	Obtención de la autorización	73
10.1.2.	Obtención del token de acceso	73
10.1.3.	Consulta de los datos de usuario.....	74
10.2.	Servicio de firma.....	74
10.2.1.	Obtención del token OAuth.....	74

10.2.2.	Creación del proceso de firma.....	75
10.2.3.	Ejecución del proceso de firma	76
10.2.4.	Obtención del documento firmado	76
10.2.5.	Eliminación del proceso de firma	76
11.	Ejemplos de integración mediante Java	77
11.1.	Servicio de autenticación	77
11.2.	Servicio de firma.....	78
11.2.1.	Firma PAdES	78
11.2.2.	Firma XAdES Enveloped	84
11.2.3.	Firma XAdES Externally Detached EPES.....	95
11.2.4.	Firma XAdES Externally Detached a partir de un hash.....	98
11.2.5.	Firma CAdES Attached.....	100
11.2.6.	Firma lotes PAdES con diferente apariencia en la firma visible	102
11.2.7.	Firma lotes XAdES Externally Detached a partir de un hash	107
11.2.8.	Firma lotes combinando diferentes perfiles de firma (PAdES + XAdES).....	102
12.	Ejemplos de verificación de firmas mediante Zain	77
12.1.	Firma XAdES Enveloping con elemento Manifest.....	77
12.2.	Firma XAdES Enveloping con varios elementos Manifest	78
13.	Referencias.....	80

1. Introducción

Giltza es la plataforma técnica sobre la que se construye el portal de identificación y firma electrónica de las administraciones públicas de Euskadi. A su vez, el concepto de Giltza se asienta sobre dos plataformas técnicas diferenciadas con idénticas capacidades, en las que se han desarrollado diferentes proveedores de identidad, uno dedicado al ámbito ciudadanía y otro dedicado al ámbito profesional.



Las diferentes URL de Giltza pueden verse en el apartado [3. Endpoints de Giltza](#).

1.1. Objeto del documento

El objetivo de este documento es detallar todos los elementos necesarios para integrar una aplicación con los servicios de autenticación, firma y consulta de operaciones/ eventos proporcionados por Giltza. A este efecto, se detallan las APIs más relevantes de Giltza:

- *API de autorización OAuth 2.0:* para obtener los tokens OAuth con los que acreditar que disponen de la autorización pertinente para acceder a los recursos que proporciona la plataforma (datos de identidad y de los procesos de autenticación de los usuarios, claves de firma, etc.)
- *API de información de usuario:* para obtener información sobre la identidad y los procesos de autenticación de los usuarios que han sido autenticados por Giltza en el curso de un flujo de autorización OAuth.
- *API de firma de documentos:* para realizar firmas electrónicas avanzadas de documentos (firmas PAdES, XAdES y CAdES).
- *API de consulta de operaciones y eventos:* posibilidad de consultar las operaciones principales que realizan los usuarios a través de los servicios de Giltza, así como todos los eventos que se registran en el log.

Además, se realizará una descripción acerca de los siguientes elementos para una correcta integración con Giltza:

- Flujo de autenticación “*step-up*” para conseguir aumentar el nivel de aseguramiento o calidad (LoA).
- Utilización de la opción de federación que Giltza tiene integrada, de forma que sólo sea necesario autenticarse una vez para poder hacer uso de todos los servicios proporcionados por la infraestructura de Giltza.
- Status de las respuestas HTTP (códigos de error).
- Integración mediante SAML 2.0.
- Ejemplos de integración a través del cliente HTTP denominado cURL, permitiendo construir mensajes y enviar peticiones REST a Giltza.
- Ejemplos de integración con Giltza mediante el lenguaje de programación Java para utilizar el servicio de autenticación con OAuth 2.0, el servicio de firma y el servicio de consulta de operaciones y eventos.

NOTA: A lo largo del documento se utilizan indistintamente las expresiones *Proveedor de Servicios*, *Relying Party (RP)*, *aplicación cliente* o *aplicación* para referirse a la aplicación web o móvil nativa que utilizará los servicios de Giltza.

2. Autorización para utilizar las APIs

Para invocar a los servicios REST de Giltza, las aplicaciones deben utilizar el siguiente mecanismo de autorización disponible:

- *Tokens de acceso OAuth*: las aplicaciones cliente pueden utilizar, para obtener tokens de acceso de un servidor de autorización Giltza, dos tipos de flujos OAuth2.0:
 - *Authorization Code Grant*: cuando se usa este flujo el usuario siempre está presente, y debe autenticarse previamente en Giltza para dar su consentimiento. Este consentimiento puede ser explícito o implícito por la propia autenticación del usuario.
 - *Client Credentials Grant*: la aplicación sólo necesita autenticarse como sí misma ante Giltza para obtener el token de acceso y este token obtenido no está asociado a ningún usuario en particular.

3. Endpoints de Giltza

Las URL o endpoints disponibles para Giltza se muestran a continuación:

IdP	Entorno	Valor
Giltza	Desarrollo	https://eidasdes.izenpe.com:8082/
	Producción	https://eidas.izenpe.com/
Giltza Profesional	Desarrollo	https://eidas-profdes.izenpe.eus:8082/
	Producción	https://eidas-prof.izenpe.eus/

4. Registro de una aplicación

Antes de comenzar la integración de una nueva aplicación en la plataforma, es necesario que el integrador esté en posesión de las credenciales específicas para su aplicación. Para ello, **izenpe** facilitará la siguiente información (strings) que será usada en diferentes invocaciones a los servicios:

- `client_id`: identificador de la aplicación cliente dada de alta en Giltza.
- `client_secret`: contraseña asociada a la aplicación cliente.

A partir de estos elementos, será necesario calcular el valor de la API Key mediante el algoritmo:

```
base64(url_encode(utf8(client_id)) ':' url_encode(utf8(client_secret)))
```

La aplicación deberá utilizar la *API Key* como credencial de autenticación en todas las peticiones que dirija al servicio de autenticación y autorización para obtener un token de acceso a cualquiera de los recursos protegidos de la plataforma. Para ello, tal como establece OAuth 2.0, se indicará la *API Key* en la cabecera *Authorization* de dichas peticiones de acuerdo con la siguiente expresión:

```
Authorization: Basic <API-Key>
```

4.1. URIs de redirección permitidas

Los integradores de Giltza tienen la obligación de informar de las URIs de redirección o callback URIs correspondientes a los procesos de autenticación, con el objetivo de cumplir con el **RFC6749 “The OAuth 2.0 Authorization Framework”**, concretamente para el punto **3.1.2. Redirection Endpoint**, que indica lo siguiente:

*“Después de completar su interacción con el propietario del recurso, el servidor de autorización redirige el agente de usuario del propietario del recurso de vuelta al cliente. El servidor de autorización redirige el agente de usuario al **punto de redirección del cliente previamente establecido con el servidor de autorización durante el proceso de registro del cliente o al realizar la solicitud de autorización.***

*La URI del punto de redirección **DEBE** ser una **URI absoluta** como se define en la [RFC3986] Sección 4.3. La URI del punto de redirección **PUEDE** incluir un componente de consulta formateado como "application/x-www-form-urlencoded" (según el Apéndice B) ([RFC3986] Sección 3.4), el cual **DEBE** ser conservado al agregar parámetros de consulta adicionales. La URI del punto de redirección **NO DEBE** incluir un componente de fragmento.”*

Izenpe deberá estar informada de todas las URIs de redirección utilizadas por los integradores, tanto para el entorno de Giltza Desarrollo como para Giltza Producción.

Por otra parte, la información relativa a las URIs de redirección deberá ser enviada al siguiente correo electrónico, junto con el identificador de la aplicación cliente dada de alta en Giltza (`client_id`): atecnica@izenpe.eus.

5. Integración con el servicio de autenticación OAuth 2.0

5.1. Kit de integración con el servicio de autenticación

Este kit de integración con el servicio de autenticación comprende los elementos para integrar las distintas aplicaciones con los servicios de autenticación proporcionados por el sistema Giltza.

Cuando la integración con el servicio de autenticación de Giltza se realice mediante el protocolo OAuth 2.0, los pasos a seguir son:

1. Obtención de la autorización.
2. Obtención del token de acceso.
3. Consulta de los datos de usuario.

Independientemente del mecanismo de autenticación que se quiera emplear, cada proceso necesita el intercambio de tres peticiones de servicio, cuyos mensajes se definen en OAuth 2.0 (RFC 6749) y en OpenID Connect 1.0:

1. Obtención de la autorización (petición *OAuth 2.0 Authorization Request*), dónde se produce la autenticación. El endpoint del servicio es:

<https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe>

- Obtención del token de acceso (petición *OAuth 2.0 Token Request*). El endpoint asociado a este servicio es:

https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe/token

- Consulta de los datos de usuario (petición *OpenID Connect UserInfo Request*). El endpoint es el siguiente:

https://eidas.izenpe.com/trustedx-resources/openid/v1/users/me

5.1.1. Obtención de la autorización

Realiza una petición de autorización que inicia un flujo de autorización OAuth 2.0. Los mensajes de esta operación no son intercambiados de forma directa entre la aplicación cliente y Giltza, sino indirectamente a través del navegador del usuario, mediante redirecciones del navegador.

Esta operación permite a una aplicación obtener autorización del usuario para acceder a un recurso de su propiedad. La autorización del usuario se obtiene en forma de una credencial denominada *código de autorización*. El código de autorización viaja a través del navegador en forma de respuesta de la operación.

El código de autorización se obtiene mediante la petición y la respuesta descritas a continuación.

5.1.1.1 Petición

La petición de autorización se hace mediante un mensaje HTTP GET al endpoint de autorización, directamente o por medio de una respuesta de redirección al navegador del usuario similar a:

```
https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe?
response_type=code&
client_id=<client_id>&
redirect_uri=<redirect_uri>&
scope=<scope>& acr_values=<acr>&
state=<random>&
prompt=<prompt>&
ui_locales=<ui_locales>&
login_hint=<login_hint>
```

Los parámetros de la petición son:

Nombre	Valor	Presencia	Descripción
response_type	code	Obligatorio	Valor definido por OAuth 2.0 para obtener un código de autorización.
client_id	<client_id>	Obligatorio	Identificador de la aplicación registrada en Giltza.
redirect_uri	<redirect_uri>	Obligatorio	URI de redirección a la aplicación donde ésta espera recibir la respuesta de Giltza con el código de autorización.

scope	<scope>	Opcional	Lista de valores, separados por espacio, que define sobre qué elementos solicita autorización la aplicación.
acr_values	<acr>	Opcional	Nivel o flujo de autenticación solicitado.
state	<random>	Recomendado	Valor aleatorio que identifica a la petición y evita los ataques CSRF.
prompt	<prompt>	Opcional	Sólo admite los valores login y none.
ui_locales	<ui_locales>	Opcional	Lista de etiquetas de idiomas según RFC 5646, separadas por espacios.
login_hint	<login_hint>	Opcional	Identificador de usuario sugerido que puede ser usado si es necesario autenticar al usuario.

<scope>

Los valores disponibles para el parámetro *scope* quedan recogidos en el apartado *Atributos de usuario en OAuth2.0*. Si no aparece este parámetro se aplicará el *scopeurn:izenpe:identity:global*.

<acr_values>

Los flujos de autenticación definidos en el entorno de Giltza ciudadano basándonos en la clasificación de niveles de eIDAS son:

- Giltza Ciudadano



The screenshot shows the Izenpe authentication interface. At the top, there is the Izenpe logo and the text "Identificación electrónica de Euskadi". Below this, it says "Izenpe solicita su autenticación." and "Seleccione cuál de los siguientes medios de identificación desea utilizar:". There are five options listed in a vertical list:

- BAK**: DNI/NIE/PASAPORTE y contraseña
- BAKQ**: DNI/NIE, contraseña y coordenadas; DNI/NIE, contraseña y código SMS
- Certificados digitales**
- izenpe mobile**
- Cl@ve**

Flujo de Autenticación	Nivel	Identificador
Bak	Bajo	urn:safelayer:twspolicies:authentication:flow:bak
BakQ	Sustancial	urn:safelayer:twspolicies:authentication:flow:bakq
Izenpe Mobile	Sustancial	urn:safelayer:twspolicies:authentication:flow:izmobileid:citizen
Cl@ve 2 [PIN 24H]	Sustancial	urn:safelayer:twspolicies:authentication:flow:clave:aeat
Cl@ve 2 [Clave Permanente]	Bajo	urn:safelayer:twspolicies:authentication:flow:clave:ss
Cl@ve 2 [PIN 24H + Clave Permanente]	Sustancial	urn:safelayer:twspolicies:authentication:flow:clave
Certificado	Alto	urn:safelayer:twspolicies:authentication:flow:cert

- Giltza Profesional



Flujo de Autenticación	Nivel	Identificador
Giltza Profesional	Sustancial	urn:izenpe:authentication:flow:bak_otp
Izenpe Mobile	Sustancial	urn:safelayer:twspolicies:authentication:flow:izmobileid:prof
Certificado	Alto	urn:safelayer:twspolicies:authentication:flow:cert

- Giltza Ciudadano + Giltza Profesional (**federación de identidades**):

Hay ocasiones en el que el universo en el que se desarrolla la aplicación no diferencia entre el ámbito ciudadano y ámbito profesional y la persona usuaria de la misma puede autenticarse haciendo uso de flujos de autenticación de Giltza Ciudadano o de Giltza Profesional.

Para hacer esta funcionalidad plausible, existen dos opciones:

1. El desarrollador crea un frontal con los diferentes flujos que le interesa y el mismo se encarga de llamar en cada caso a la plataforma concreta y flujo adecuado (sería el caso que utiliza la sede de Gipuzkoa).
2. Se utiliza una de las características técnicas de la propia plataforma Giltza como es la federación con proveedores de identidad externos.

Izenpe ha federado los flujos de autenticación propios de Giltza Profesional en Giltza Ciudadano, de tal forma que desde Giltza Ciudadano se pueda trabajar con ellos, en esta situación los identificadores que debemos utilizar para los flujos propios de Giltza Profesional son los que se muestran en la tabla adjunta:



Flujo de Autenticación	Nivel	Identificador
Giltza Profesional	Sustancial	urn:safelayer:tws:policies:authentication:flow:giltza:profesional

Izenpe Mobile	Sustancial	urn:safelayer:twspolicies:authentication:flow:izmobileid:citizen_prof
---------------	------------	---

En ambos Giltza, en el proceso de petición, la aplicación puede indicar los flujos que desea incorporar (en ese caso tendrá que utilizar los identificadores que se han indicado en los apartados anteriores) o el nivel mínimo de los flujos que desea incorporar. En el caso que elija definir un nivel, en “la botonera” se incluirán todos los flujos asociados **por defecto** a ese nivel y superiores.

Los identificadores asociados a cada nivel son los que se muestran en la tabla adjunta:

Nivel requerido	Identificador
Bajo	urn:safelayer:twspolicies:authentication:level:low
Sustancial	urn:safelayer:twspolicies:authentication:level:medium
Alto	urn:safelayer:twspolicies:authentication:level:high

Cada uno de los niveles implica los siguientes flujos por defecto:

- Nivel Bajo: Bak, BakQ y certificado digital, si se quiere añadir cualquier otro hay que invocarlo directamente.
- Nivel Sustancial: BakQ y Certificados digitales, si se quiere añadir cualquier otro hay que invocarlo directamente.
- Nivel Alto: certificado digital, si se quiere añadir cualquier otro hay que invocarlo directamente.

Además, habrá que tener en cuenta los siguientes puntos:

- Si no se especifica el parámetro *acr_values* o está vacío, el sistema mostrará un selector con todos los flujos disponibles, y el usuario deberá elegir uno de ellos.
- Si en el parámetro *acr_values* se especifica un valor de nivel de aseguramiento concreto, el sistema mostrará un selector con los flujos que están clasificados en dicho nivel, y el usuario deberá elegir uno de ellos. Si sólo existe un mecanismo de dicho nivel, se activará dicho mecanismo directamente.
- Si en este parámetro se incluye un valor de flujo concreto, el sistema intentará aplicar de forma directa el flujo identificado por dicho valor. Para indicar un conjunto de flujos de autenticación concretos seleccionables por el usuario, se deben indicar los identificadores en formato URN de los flujos, separados por la barra vertical '|'. El orden no es relevante. Posibles ejemplos son los siguientes:

acr_values=urn:safelayer:twspolicies:authentication:level:low|urn:safelayer:twspolicies:authentication:flow:giltza:profesional

acr_values=urn:safelayer:twspolicies:authentication:flow:bak|urn:safelayer:twspolicies:authentication:flow:cert

A continuación, se muestran distintos ejemplos de esta petición según el resultado que se desea obtener:

Autenticación con BakQQ y acceso al perfil del usuario y suemail

```
https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe?
  response_type=code&
  client_id=<client_id>&
  redirect_uri=<redirect_uri>&
  scope=profile email&
  acr_values=urn:safelayer:twspolicies:authentication:flow:bakq& state=<random>
```

Autenticación con selector que sólo muestra mecanismos de nivel medio (y superior) y solicita acceso al perfil del usuario

```
https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe?
  response_type=code&
  client_id=<client_id>&
  redirect_uri=<redirect_uri>&
  scope=profile&
  acr_values=urn:safelayer:twspolicies:authentication:level:medium&state=<random>
```

Autenticación con selector (sin restricciones) y acceso al perfil del usuario

```
https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe?
  response_type=code&
  client_id=<client_id>&
  redirect_uri=<redirect_uri>&
  scope=profile&state=<random>
```

<prompt>

Se pueden distinguir entre varios valores posibles:

- El valor *login* sirve para desactivar el SSO(*Single-Sign-On*).
- El valor *none* sirve para indicar que no puede haber interacción con el usuario (ni para autenticarlo, ni para obtener su consentimiento).

Se puede usar por ejemplo en el caso de que una RP pueda forzar que un proceso de autenticación falle si se requiere interacción con el usuario y puede devolver 3 causas de error distintas:

- *login_required*: no hay login del usuario.
- *interaction_required*: hay login previo, pero no se puede aplicar SSO(*Single-Sign-On*).
- *consent_required*: hay login y se puede aplicar SSO (*Single-Sign-On*), pero el usuario debe autorizar explícitamente el consentimiento.

<ui_locales>

Este parámetro no es estándar y sirve para indicar una lista de idiomas ordenada por preferencia. Giltza utilizará esta lista para seleccionar el idioma en el que deberá mostrar la interfaz gráfica con la que se autentique al usuario y se obtenga su consentimiento.

La interfaz se mostrará internacionalizada según la primera etiqueta de la lista cuyo idioma base esté

soportado por Giltza o haya sido personalizado. Los valores hacen referencia a los códigos definidos en el standard ISO 639. Los idiomas disponibles son:

Idioma	Etiqueta
Español	es
Euskera	eu
Inglés	en

<login_hint>

Se utiliza cuando el aplicativo (RP) es capaz de saber qué usuario se está autenticando, dando opción a utilizar este parámetro como sugerencia en Giltza. De esta forma, el campo *DNI* del formulario de autenticación de Giltza aparecerá completo.

Para hacer uso de este parámetro, bastará con incluir en el inicio del flujo de autenticación OAuth 2.0 el siguiente código:

```
[...]
.setParameter("login_hint", "<dni>")
[...]
```

5.1.1.2 Respuesta

El mensaje de respuesta tendrá el siguiente formato:

```
HTTP/1.1 302 Found
Location: <redirect_uri>?code=<code>&state=<random>
```

La aplicación recibe un mensaje HTTP GET en la uri de redirección *<redirect_uri>* con los siguientes parámetros:

- *code*: código de autorización concedido.
- *state*: valor aleatorio que identifica a la petición y evita los ataques CSRF (la aplicación deberá comprobar que el valor coincide con el enviado en la petición).

5.1.2. Obtención del token de acceso

Mediante este paso, el token de acceso obtenido acredita la autorización que un usuario de un determinado dominio de identidad concede a la aplicación que efectúa la llamada para que acceda a determinados recursos o servicios en su nombre (datos de identidad, identidades de firma, etc.).

El token de acceso se obtiene mediante la petición y la respuesta descritas a continuación.

5.1.2.1 Petición

Para solicitar el token de acceso, la aplicación debe mandar una petición HTTP POST con el

siguiente contenido:

```
POST /trustedx-authserver/oauth/izenpe/token HTTP/1.1
Host: eidas.izenpe.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Basic <api_key>

grant_type=authorization_code&
redirect_uri=<redirect_uri>&
code=<code>
```

Los parámetros de la petición son los siguientes:

Nombre	Valor	Presencia	Descripción
api_key	<api_key>	Obligatorio	API Key obtenido en el registro de la aplicación.
grant_type	authorization_code	Obligatorio	Valor definido por OAuth 2.0 para intercambiar un código de autorización por un token de acceso.
redirect_uri	<redirect_uri>	Opcional/Obligatorio	Obligatorio si la petición de autorización incluía el parámetro <i>redirect_uri</i> . URI de redirección especificada en la petición de autorización.
code	<code>	Obligatorio	Valor del parámetro code obtenido en la respuesta del mensaje anterior.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
```

5.1.2.2 Respuesta

```
{
  "scope": "<scope>",
  "expires_in": 120,
  "token_type": "Bearer",
  "access_token": "<access_token>"
}
```

Los parámetros de la respuesta de ejemplo son:

- *scope*: el mismo que se le solicita en la primera petición de autorización.
- *expires_in*: segundos durante los que el token de acceso es válido.
- *token_type*: tipo de token de acceso. Siempre tendrá el valor *Bearer*.
- *access_token*: el token que la aplicación deberá incluir en peticiones a los servicios en nombre

del usuario autenticado.

5.1.3. Consulta de los datos de usuario

Se proporciona información de identidad y del proceso de autenticación del usuario asociado a una autorización OAuth 2.0.

Los datos de usuario se consultan mediante la petición y la respuesta descritas a continuación.

5.1.3.1 Petición

Para solicitar datos del usuario relacionado con un token de acceso, la aplicación debe mandar una petición HTTP GET con el token recibido en el mensaje anterior:

```
GET /trustedx-resources/openid/v1/users/me HTTP/1.1
Host: eidas.izenpe.com
Authorization: Bearer <access_token>
```

5.1.3.2 Respuesta

Como respuesta, la obtención obtendrá un JSON con los detalles del usuario asociados al *scope* solicitado. Un ejemplo sería el siguiente mensaje:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache

{
  "sub" : "<identificador>",
  "domain" : "izenpe",
  "acr" : "<acr>",
  "email" : "<email>",
  "name" : "<name>",
  "dni" : "<dni>",
}
```

Los parámetros correspondientes al ejemplo son los siguientes:

Nombre	Valor	Descripción
sub	<identificador>	Identificador del usuario.
domain	izenpe	Dominio de identidad.
acr	<acr>	Flujo de autenticación superado o nivel de seguridad alcanzado por el usuario.
email	<email>	Correo electrónico del usuario.
name	<name>	Nombre completo del usuario.
dni	<dni>	Número de DNI del usuario.

5.1.4 Atributos de usuario en OAuth 2.0

La tabla siguiente detalla la información del usuario autenticado que puede obtener una aplicación OAuth 2.0.

Atributo	Tipo	Presencia	Descripción
sub	String	Siempre	Identificador de usuario (provisionalmente se recomienda no utilizar este atributo para mapear la identidad de Giltza a una cuenta local de la aplicación).
domain	String	Siempre	Dominio de identidad (izenpe/ funcionarios).
acr	String	Siempre	Flujo de autenticación superado o nivel de seguridad (LoA) alcanzado por el usuario.
amr	Array	Siempre	Mecanismos de autenticación que el usuario ha superado con éxito a lo largo de la sesión en Giltza.
name	String	Opcional	Nombre y apellidos del usuario.
dni	String	Opcional	Número de DNI (mientras no haya vinculación de cuentas no es posible obtenerlo si el usuario sólo se autentica con su identidad de Facebook).
given_name	String	Opcional	Nombre del usuario.
family_name	String	Opcional	Apellidos del usuario.
surname1	String	Opcional	Primer apellido del usuario.
surname2	String	Opcional	Segundo apellido del usuario.
country	String	Opcional	País emisor del número del DNI, en formato ISO 3166-1 alpha-2 d.
birthdate	String	Opcional	Fecha de nacimiento del usuario, en formato YYYY-MM-DD.
email	String	Opcional	Dirección de correo electrónico del usuario.
issuer	String	Opcional	Emisor de la credencial.
person_status	String	Opcional	Tipo de persona representada por el usuario: física (PF) o jurídica (PJ).
organization	String	Opcional	Razón social de la entidad jurídica autenticada.
cif	String	Opcional	Número de CIF de la entidad jurídica representada por el usuario.
not_before	String	Opcional	Fecha inicial de validez de la credencial.
not_after	String	Opcional	Fecha final de validez de la credencial.
policy_identifier	String	Opcional	OID del certificado.
external_info	Compuesto	Opcional	Información procedente de terceros (Facebook, federación, etc.)
serial_number	String	Opcional	Número de serie del certificado de usuario.

key_usage	String	Opcional	Usos de clave.
subject	String	Opcional	Subject del certificado de usuario.
tls	String	Opcional	Presencia del certificado en la TSL. Posibles valores: "S", "N"
register_type	String	Opcional	Tipo de proceso de registro realizado. Posibles valores: "1"
fea_owner	String	Opcional	Indica si el usuario autenticado tiene emitido BakQ ("true") o no ("false")
país	String	Opcional	País emisor del número del DNI, en formato ISO 3166-1 alpha-2 d
firmaCualificada	String	Opcional	Indica si el certificado es cualificado (S) o no (N). Posibles valores: "EMPTY", "S", "N", "N/A" (No Aplica).
tipoAFirma	String	Opcional	Categorización del certificado según @firma. Posibles valores: "EMPTY", "N/A" (No Aplica) o "<dígito>".
legalPersonSemanticsIdentifier	String	Opcional	Identificador asociado al certificado de persona jurídica.
naturalPersonSemanticsIdentifier	String	Opcional	Identificador asociado al certificado de persona física.
perfil	String	Opcional	Indica si el usuario pertenece al perfil de profesional (0) o funcionario (1).

La aplicación OAuth 2.0 debe incluir los scopes adecuados en la petición de autorización para obtener determinados conjuntos de atributos de identidad. A los atributos incluidos en cada uno de los *scopes*, hay que incluir los siguientes que siempre aparecen: *sub*, *domain*, *acr* y *amr*.

Scope	Atributos
profile	given_name, family_name, name, birthdate
email	email
urn:izenpe:identity:global	given_name, family_name, name, birthdate, surname1, surname2, dni, country, issuer, policy_identifier, organization, cif, not_before, not_after, serial_number, key_usage, subject, tls, pais, email, person_status, register_type, firmaCualificada, tipoAFirma, legalPersonSemanticsIdentifier, naturalPersonSemanticsIdentifier
urn:izenpe:zain:xml	zain_xml
urn:izenpe:fea:properties	fea_owner No aplicable a Giltza Profesional

urn:safelayer:eid:sign:identity:profile	Atributo <i>sign_identities</i> , que contiene la información sobre las identidades de firma electrónica del usuario
urn:safelayer:eid:authn_details	Atributo <i>authn_details</i> , que contiene: <i>authnFlow</i> , <i>directSso</i> , <i>authnLevel</i> , <i>targetAcr</i> , <i>bak_authenticated</i> Para ver los posibles atributos correspondientes a Izenpe Mobile (a parte de los mencionados), continuar al apartado <i><authn_details> en Izenpe Mobile</i>
urn:safelayer:eid:external_info	Atributo <i>external_info</i> , que puede contener un número variable de atributos dependiendo de la información procedente de terceros (proveedores de identidad federados u otras fuentes de identidad externas)

NOTA: Debido a que es posible que el valor de algunos atributos sea *EMPTY* cuando no se disponga información al respecto con el mecanismo de autenticación seleccionado, puede ocurrir que tras hacer una re-autenticación con diferentes mecanismos el valor de alguno de los atributos sea un array con los valores acumulados.

<authn_details> en Izenpe Mobile

Giltza establecerá los siguientes atributos posibles dentro de los detalles de autenticación, una vez que el usuario se haya autenticado correctamente:

Nombre	Descripción
deviceTag	Id. del dispositivo
deviceFingerprint	Huella del dispositivo
deviceTrackerTheftAlert	Detectado robo de la cookie del dispositivo
contextAnalysisPolicy	Política utilizada para realizar el análisis
contextAnalysisAdvice	Resultado (la recomendación) del análisis de contexto
keystrokelsEnabled	Dinámica de tecleo habilitada
keystrokeNumberOfKeys	Número de caracteres de las credenciales
keystrokeSamplesAvailable	Muestra de tecleo disponible
keystrokeNumberOfKeySets	Número de conjuntos de caracteres de las credenciales
keystrokeSamplesConsistent	Muestra de tecleo consistente
keystrokeResult	Resultado de la dinámica de tecleo

devicePreferredLanguage	Idioma predefinido del dispositivo
deviceIsIdentifiable	Dispositivo identificado
deviceOperatingSystem	Sistema operativo del dispositivo
deviceIsRegistered	Dispositivo registrado
deviceBrowserVersion	Versión del navegador del dispositivo
deviceFingerprintQuality	Calidad de la huella del dispositivo
deviceBrowser	Navegador del dispositivo
deviceUserAgent	Cabecera HTTP User-Agent
ipAddress	Dirección IP
userIsNew	Usuario nuevo
mobileIdAuthnMode	Modo de autenticación
locationCountryName	Nombre del país desde el que se realiza el acceso
locationCountryCode	Código del país desde el que se realiza el acceso
locationLongitude	Longitud de las coordenadas
locationLatitude	Latitud de las coordenadas
riskThreshold	Umbral de riesgo
riskScore	Valoración de riesgo
factor.*.flag	Factores que deben validarse con éxito
factor.*.result	Factores que se han validado con éxito y cuáles no

5.1.5 Logout

Actualmente, en Giltza se puede forzar o finalizar la sesión de un usuario de las siguientes maneras, dependiendo del mecanismo de autenticación que se haya utilizado.

Mecanismo	URL
Cualquiera (menos Clave 2)	<a href="https://<giltza_url>:<giltza_port>/trustedx-authserver/izenpe/logout?redirect_uri=<redirect_uri>">https://<giltza_url>:<giltza_port>/trustedx-authserver/izenpe/logout?redirect_uri=<redirect_uri>
Clave 2	<a href="https://<giltza_url>:<giltza_port>/clave-authn-saml2/logout?redirect_uri=<redirect_uri>">https://<giltza_url>:<giltza_port>/clave-authn-saml2/logout?redirect_uri=<redirect_uri>

Si el usuario se ha autenticado en Clave 2 mediante Giltza, la recomendación es realizar un *logout* en

ambos IdP (Identity Provider), de forma que la sesión finalice en los dos ámbitos: Giltza y Clave 2. Por tanto, habrá que invocar a las dos URL para asegurarse de que la sesión del usuario finaliza correctamente.

5.2. Flujo step-up

El objetivo de la utilización de este tipo de flujo es realizar una re-autenticación para elevar el nivel de garantías (LoA) que el proveedor de servicios tiene de la identidad de los usuarios que han iniciado una sesión autenticándose con un mecanismo que ofrece pocas garantías (mediante contraseña) y que comienza cuando el usuario hace alguna petición que el proveedor de servicios sólo puede atender si conoce su identidad con un nivel de garantías mayor (step-up).

Por lo tanto, para poder elevar o controlar este nivel de garantías mediante el RP y Giltza, es necesario seguir estos pasos:

1. A la hora de iniciar el flujo OAuth 2.0, el valor del parámetro *acr_values* tendrá que ser el correspondiente a BakQ:

urn:safelayer:twspolicies:authentication:flow:bak:extended

2. En el *callback* de la autenticación, habrá que realizar las siguientes comprobaciones:

- Si el atributo JSON del UserInfo denominado *fea_owner* tiene el valor “true” y el atributo *authn_details.authnFlow* tiene el mismo valor que el parámetro *acr_values* indicado en el primer punto, entonces, se podrá realizar una re-autenticación mediante la redirección a otra clase controladora que inicie un nuevo flujo de OAuth 2.0. Para este caso, el valor del parámetro *acr_values* será:

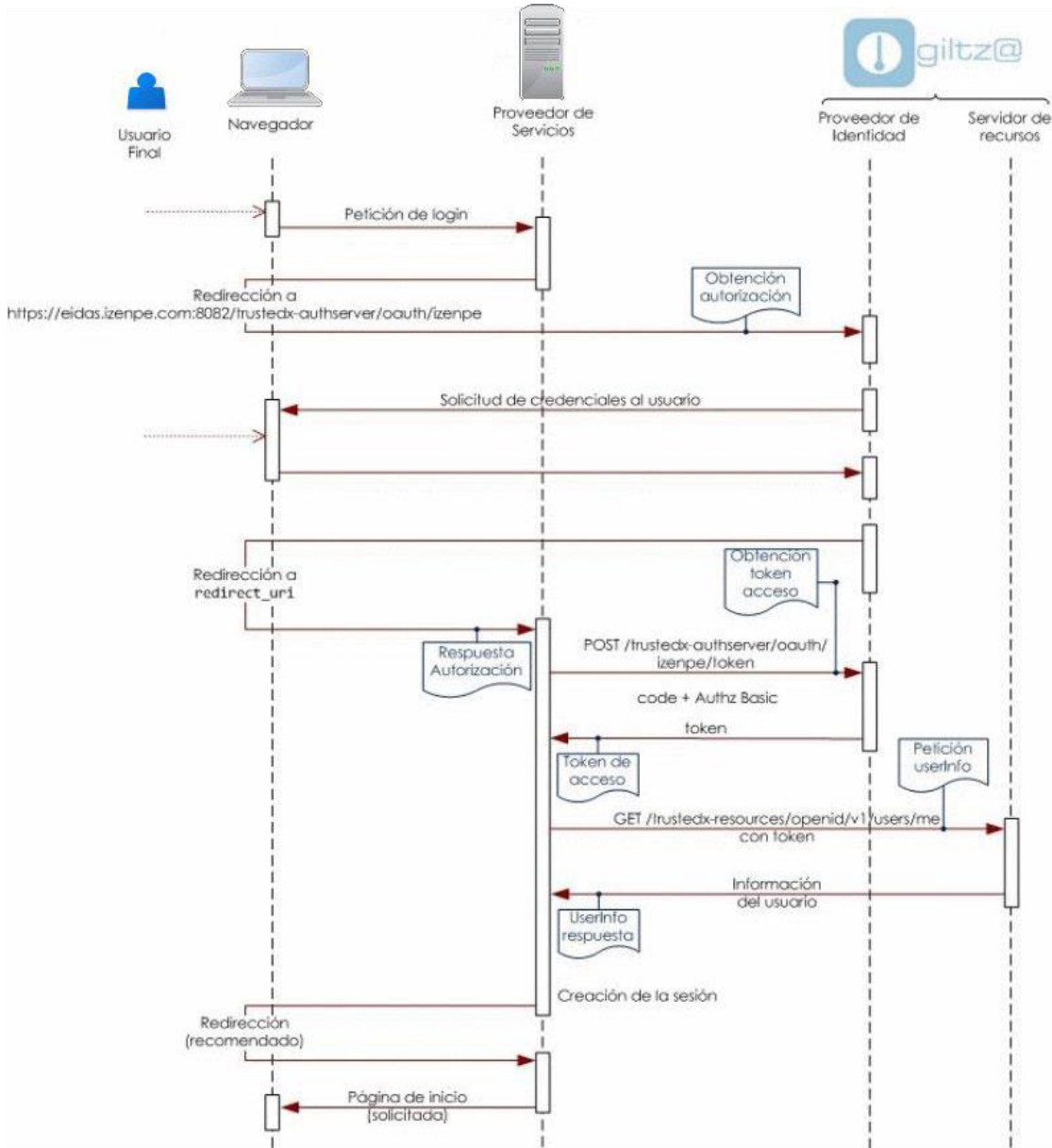
urn:safelayer:twspolicies:authentication:flow:fea:stepup

De esta forma, y una vez realizada la segunda autenticación correctamente, se realizará un salto en el nivel de garantías, pasando de nivel bajo a medio.

- Si el atributo JSON denominado *fea_owner* tiene el valor “false” o el atributo *authn_details.authnFlow* tiene un valor diferente que el parámetro *acr_values* indicado en el primer punto, entonces se realiza la autenticación como si de un flujo cualquiera se tratara.

5.3. Diagrama de flujo para autenticación con OAuth2.0

El siguiente diagrama muestra el flujo de mensajes definidos para realizar una autenticación OAuth mediante Giltza.



6. Integración con el servicio de autenticación SAML 2.0

Una manera de integrar aplicaciones con Giltza es mediante SAML 2.0. Giltza puede actuar como un proveedor de identidad SAML 2.0 en el que las aplicaciones, actuando como proveedores de servicios, pueden delegar la autenticación/SSO de los usuarios.

Giltza implementa el perfil *Web Browser SSO* de SAML 2.0 para emitir aserciones, pero con algunas limitaciones. Éste actúa como la entidad *Identity Provider* (IdP) definida en SAML 2.0, mientras que el rol *Service Provider* (SP) corresponde a la aplicación (o aplicaciones) que se desea integrar con Giltza en modo Software as a Service.

6.1. Kit de integración con el servicio de autenticación

Este kit de integración con el servicio de autenticación comprende los elementos para integrar las distintas aplicaciones con los servicios de autenticación proporcionados por el sistema Giltza.

Cuando la integración con el servicio de autenticación de Giltza se realice mediante el estándar SAML 2.0, los pasos a seguir son:

1. Petición de autenticación HTTP
2. Respuesta de autenticación HTTP

6.1.1. Petición de autenticación HTTP

Para iniciar el flujo *Web Browser SSO*, el proveedor de servicios construye una petición de autenticación y la envía a Giltza a través del navegador del usuario, mediante el *binding* HTTP Redirect o HTTP POST.

La petición debe enviarse siempre por TLS. Los parámetros que incluir en la petición son:

Parámetro	Presencia	Descripción
SAMLRequest	Obligatorio	Representación XML de un mensaje <i>AuthnRequest</i> , comprimida mediante el método DEFLATE y codificada en Base64
RelayState	Opcional	Datos opacos que Giltza incluirá tal cual en la respuesta de autenticación
view_type	Opcional	No estándar. Solicita a la RP que, al interactuar con el usuario para autenticarlo o re-autenticación, aplique cierto formato de visualización a las páginas web mostradas

NOTA: Todas las aplicaciones que se integran con Giltza mediante redirección del navegador (ya sea mediante OAuth 2.0 o SAML 2.0) deben hacerlo usando una URL base consistente, incluyendo el mismo nombre de dominio DNS, el mismo puerto (por defecto, 8082), y el mismo nombre de aplicación web (por defecto, *trustedx-authserver*). Los navegadores deben conectarse a Giltza siempre con la misma dirección base para que el proveedor de identidad pueda mantener las sesiones y aplicar SSO de forma consistente.

6.1.1.1. Petición

A continuación, se muestra un ejemplo de petición de autenticación enviada usando el *binding* HTTP Redirect:

```
GET /trustedx-authserver/izenpe/saml?SAMLRequest=fZLNkq...&
    RelayState=%2Fhome%2Fhome.jsp HTTP/1.1

Host: eidas.izenpe.com
Accept-Encoding: gzip, deflate
```

De los distintos elementos que puede contener el mensaje *AuthnRequest*, Giltza requiere soportar los que se indican a continuación:

Nombre	Presencia	Descripción
Version	Obligatorio	Debe tener el valor 2.0
ID	Obligatorio	Identificador único de la petición
IssueInstant	Obligatorio	Momento en que se emitió la petición, codificado en formato ISO, expresado en UTC, y sin componente de zona horaria
Issuer	Obligatorio	Identifica el emisor de la petición. Debe coincidir con el campo <i>Emisor de peticiones de autenticación</i> de alguno de los proveedores de servicios SAML 2.0 registrados en Giltza
RequestedAuthn Context	Opcional	Requisitos de autenticación solicitados (niveles mínimos y/o flujos específicos)
ForceAuthn	Opcional	Si <i>true</i> , desactiva el SSO

NOTA: Para permitir la autenticación mediante SAML 2.0, es necesario que el integrador esté en posesión de las credenciales específicas para su aplicación. Para ello, **izenpe** facilitará el valor del campo *Issuer* que será usado en las diferentes invocaciones a los servicios.

Un ejemplo de petición de autenticación SAML 2.0 puede ser el siguiente:

```
<samlp:AuthnRequest Destination="https://eidas.izenpe.com:8082/trustedx-authserver/izenpe/saml"
ID="glbjkldpjjpcahjjaomdfmlheohoeokoblmbged" IssueInstant="2018-05-08T11:18:51.309Z" Version="2.0"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">doc_sign</saml:Issuer>
  <samlp:RequestedAuthnContext>
    <saml:AuthnContextClassRef
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:safelayer:twspolicies:authentication:level:low</sam
l:AuthnContextClassRef>
  </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

6.1.2. Respuesta de autenticación HTTP

Después de autenticar o Re-autenticar al usuario (o permitirle SSO directo), Giltza generará un mensaje *Response* de SAML 2.0, como se describe a continuación, y lo enviará al proveedor de servicios a través del navegador.

Giltza usará el *binding* HTTP POST para enviar la respuesta (actualmente sólo soporta este *binding* para la respuesta). El proveedor de servicios recibirá una petición HTTP del navegador con los siguientes parámetros:

Parámetro	Presencia	Descripción
SAMLResponse	Obligatorio	Representación XML de un mensaje <i>Response</i> , comprimida mediante el método DEFLATE y codificada en Base64
RelayState	Opcional	El mismo valor que en la petición (si lo había)

6.1.2.1. Respuesta

A continuación, se muestra un ejemplo de respuesta HTTP que el navegador envía al proveedor de servicios con la respuesta de autenticación:

```
POST /saml HTTP/1.1
Host: eidas.izenpe.com
Accept-Encoding: gzip, deflate

SAMLResponse=PG5zNDpSZXNwb25zZSB4bWxuczpuczQ9...&
RelayState=%2Fhome%2Fhome.jsp
```

Giltza establecerá los siguientes elementos en el mensaje *Response*:

Nombre	Descripción
Version	Debe tener el valor 2.0
ID	Identificador único de la respuesta
IssueInstant	Momento en que se emitió la respuesta, codificado en formato ISO, expresado en UTC, y sin componente de zona horaria
InResponseTo	Identificador de la petición (mismo valor que el campo ID de la petición)

Destination	URL a la que se envía esta respuesta de autenticación. Coincidirá con el campo <i>Ubicación del Servicio consumidor de aserciones</i> configurado para el proveedor de servicios
Status	Código de estado que indica si la petición se ha podido procesar correctamente: <ul style="list-style-type: none"> • Si la petición se ha procesado con éxito, el atributo <i>StatusCode.Value</i> contendrá el valor <i>urn:oasis:names:tc:SAML:2.0:status:Success</i> • En caso contrario, los campos siguientes no estarán presentes, y este elemento contendrá información sobre el error ocurrido
Assertion	Contiene, entre otros datos, información sobre la identidad del usuario autenticado en Giltza
Signature	Firma XML del mensaje, calculada con la clave privada asociada al IdP SAML configurado en Giltza

<Assertion>

Giltza establecerá los siguientes elementos en el componente *Assertion* de la respuesta de autenticación:

Nombre	Descripción
Version	Tiene siempre el valor 2.0
ID	Identificador único de la aserción
IssueInstant	Momento en que se emitió la aserción, codificado en formato ISO, expresado en UTC, y sin componente de zona horaria
Issuer	Valor que se haya indicado en el campo <i>Emisor</i> de la configuración del Giltza
Subject	Identificador de usuario e información para la confirmación del sujeto
Conditions	Audiencia a la que va dirigida la aserción y periodo de validez
AuthnStatement	Información sobre la autenticación
AttributeStatement	Atributos de identidad del usuario

<Subject>

Giltza devolverá los siguientes datos en el elemento *Subject* de la aserción:

Nombre	Descripción
NameID	Identificador del usuario

SubjectConfirmation	<p>Uno o más elementos con información para la confirmación del sujeto:</p> <ul style="list-style-type: none"> • <i>Method</i>: fijado a <code>urn:oasis:names:tc:SAML:2.0:cm:bearer</code> • <i>SubjectConfirmationData.NotOnOrAfter</i>: coincide con el periodo de validez de la aserción • <i>SubjectConfirmationData.Recipient</i>: la URL del servicio consumidor de aserciones al que se ha enviado la información • <i>SubjectConfirmationData.InResponseTo</i>: identificador del mensaje de petición
----------------------------	--

<AuthnStatement>

El elemento *AuthnStatement* contiene información sobre cómo el usuario ha sido autenticado:

Nombre	Descripción
AuthnInstant	Momento en el que la autenticación tuvo lugar, codificado en formato ISO, expresado en UTC y sin componente de zona horaria. Actualmente, Giltza siempre incluye en este campo el instante de emisión de la aserción
AuthnContext	Nivel de autenticación con el que el usuario se encuentra autenticado, o flujo de autenticación aplicado

<AttributeStatement>

El elemento *AttributeStatement*, si lo hay, contiene atributos de identidad del usuario:

Nombre	Descripción
Name	Nombre del atributo
NameFormat	Formato del nombre del atributo <i>Name</i>
AttributeValue	Valor del atributo

Un ejemplo de respuesta de autenticación SAML 2.0 puede ser el siguiente:

```

<ns4:Response xmlns:ns4="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="http://localhost:8081/minimalist-oauth-rp/samlCallback" ID="response-
334ea32fc1a12160360dfdab954428010c8ed0c01c39c3e5171dab6187216d9b"
InResponseTo="glbjklpjjpcahjjcaomdfmlheohokobldmgbged" IssueInstant="2018-05-08T11:18:55.949Z"
Version="2.0" xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:ns2="http://www.w3.org/2001/04/xmlenc#" xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
Id="Id38131608215428078231743959350">
  <dsig:SignedInfo Id="Id115517627213625619881325838222">
    <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
    <dsig:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256" />
    <dsig:Reference Id="Id84311039611693360781964365905" URI="#response-
334ea32fc1a12160360dfdab954428010c8ed0c01c39c3e5171dab6187216d9b">
      <dsig:Transforms>
        <dsig:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
      </dsig:Transforms>
      <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
    </dsig:Reference>
  </dsig:SignedInfo>
</dsig:Signature>

```

```

<dsig:DigestValue>zXLtkcWmHcoEUMYvugYMO6P29uzu3yqFEhIwnkT62F8=</dsig:DigestValue>
  </dsig:Reference>
</dsig:SignedInfo>
<dsig:SignatureValue
Id="Id161633962220593153591548906902">J9D...QQw==</dsig:SignatureValue>
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509Certificate>MII...CfY=</dsig:X509Certificate>
    </dsig:X509Data>
  </dsig:KeyInfo>
</dsig:Signature>
<ns4:Status>
  <ns4:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</ns4:Status>
<Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID="assertion-
1b7e336ddc763ff2bf7292b597d1cf259d29aa56fe1a76a8a182b127e8fd2905" IssueInstant="2018-05-
08T11:18:55.949Z" Version="2.0">
  <Issuer>izenpe</Issuer>
  <Subject>
    <NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">CN=NOMBRE PRUEBA PRUEBA, O=IZENPE</NameID>
    <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <SubjectConfirmationData
InResponseTo="glbjkldpjpcahjjcaomdfmlheoekoblndmged" NotOnOrAfter="2018-05-08T11:23:55.949Z"
Recipient="http://localhost:8081/minimalist-oauth-rp/samlCallback"/>
    </SubjectConfirmation>
  </Subject>
  <Conditions NotBefore="2018-05-08T11:18:55.949Z" NotOnOrAfter="2018-05-
08T11:23:55.949Z">
    <AudienceRestriction>
      <Audience>doc_sign</Audience>
    </AudienceRestriction>
  </Conditions>
  <AuthnStatement AuthnInstant="2018-05-08T11:18:55.949Z">
    <AuthnContext>
      <AuthnContextClassRef>urn:safelayer:twspolicies:authentication:level:low</AuthnContextClassRef>
    </AuthnContext>
  </AuthnStatement>
  <AttributeStatement>
    <Attribute Name="dni" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
      <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">11117777Z</AttributeValue>
    </Attribute>
    <Attribute Name="name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
      <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">NOMBRE PRUEBA
PRUEBA</AttributeValue>
    </Attribute>
    <Attribute Name="given_name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
      <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">NOMBRE</AttributeValue>
    </Attribute>
    <Attribute Name="family_name"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
      <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">PRUEBA
PRUEBA</AttributeValue>
    </Attribute>
    <Attribute Name="surname1" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
      <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">PRUEBA</AttributeValue>
    </Attribute>
    <Attribute Name="surname2" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-

```

```

format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">PRUEBA</AttributeValue>
    </Attribute>
    <Attribute Name="country" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="birthdate" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">1971-01-01</AttributeValue>
    </Attribute>
    <Attribute Name="email" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">prueba@izenpe.com</AttributeValue>
    </Attribute>
    <Attribute Name="issuer" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="person_status"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">PF</AttributeValue>
    </Attribute>
    <Attribute Name="organization"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">IZENPE</AttributeValue>
    </Attribute>
    <Attribute Name="cif" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="not_before" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="not_after" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="policy_identifier"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="serial_number"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="key_usage" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
    </Attribute>
    <Attribute Name="subject" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
    <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>

```

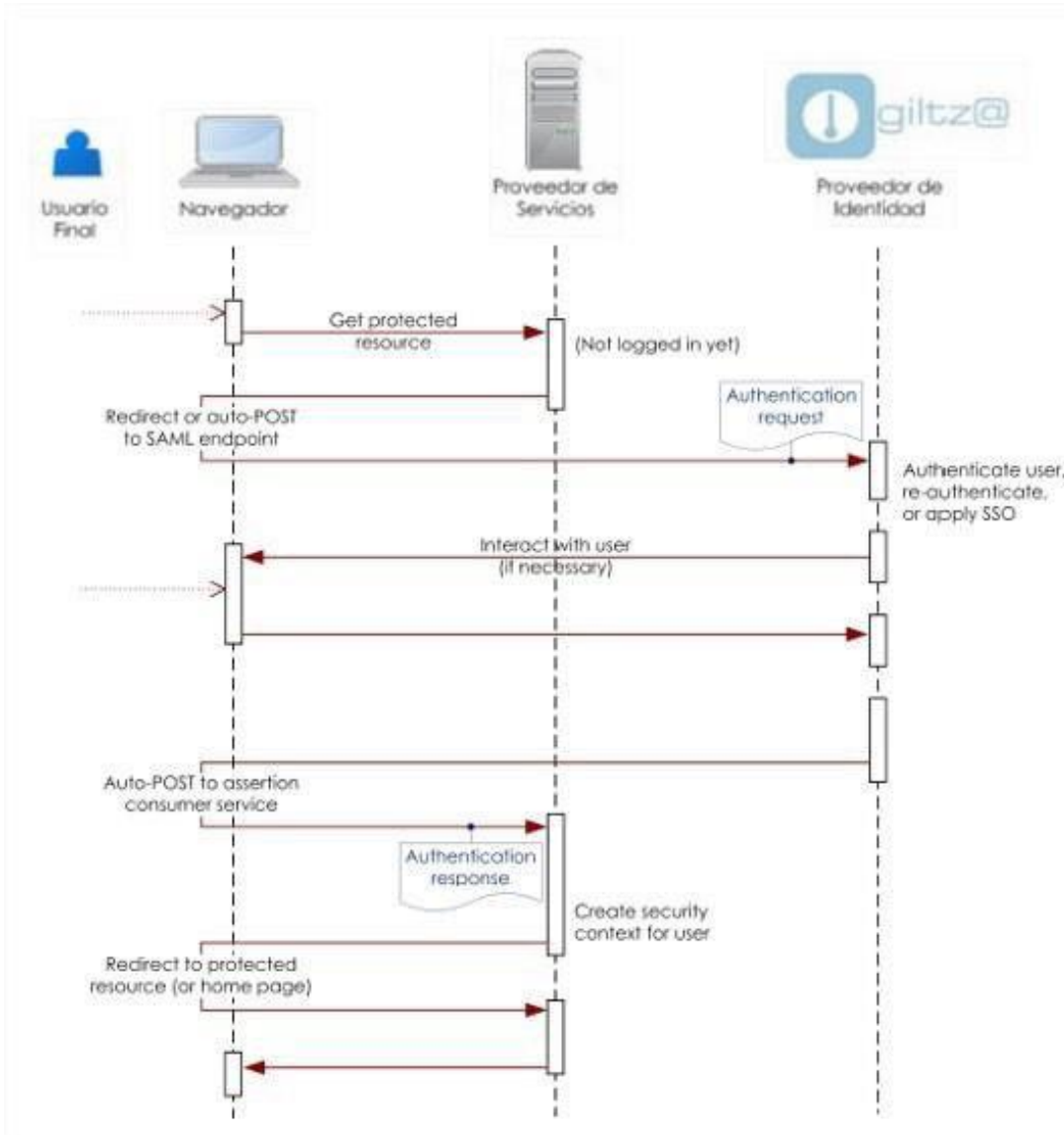
```

        </Attribute>
        <Attribute Name="ts1" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">N</AttributeValue>
        </Attribute>
        <Attribute Name="register_type"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">1</AttributeValue>
        </Attribute>
        <Attribute Name="domain" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">izenpe</AttributeValue>
        </Attribute>
        <Attribute Name="pais" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
        </Attribute>
        <Attribute Name="firmaCualificada"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
        </Attribute>
        <Attribute Name="tipoAfirma" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
        </Attribute>
        <Attribute Name="legalPersonSemanticsIdentifier"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
        </Attribute>
        <Attribute Name="naturalPersonSemanticsIdentifier"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">EMPTY</AttributeValue>
        </Attribute>
        <Attribute Name="preferencia_otp"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">sms</AttributeValue>
        </Attribute>
    </AttributeStatement>
</Assertion>
</ns4:Response>

```

6.1.3. Diagrama de flujo para autenticación con SAML 2.0

El siguiente diagrama muestra el flujo de mensajes definidos para realizar una autenticación SAML 2.0 mediante Giltza.



6.1.4. Gestión de errores

Si ocurre algún error al procesar la petición de autenticación, o durante la autenticación del usuario, Giltza enviará al proveedor de servicios una respuesta de error (en ciertos casos especiales, el error se mostrará directamente en el navegador del usuario, como se verá a continuación).

6.1.4.1. Errores sin redirección al proveedor de servicios

Si la petición de autenticación no es válida debido a uno de los siguientes motivos, Giltza mostrará un mensaje de error genérico en el navegador del usuario, indicándole que consulte a un administrador.

- La petición está mal formada (mala codificación del parámetro *SAMLRequest*, estructura XML no válida, etc.).
- El elemento *Issuer* no corresponde a ninguno de los proveedores de servicios registrados.

En estos casos, y por motivos de seguridad, no se envía ninguna respuesta al proveedor de servicios.

6.1.4.2. Errores con redirección al proveedor de servicios

Si no es posible procesar la petición de autenticación por otro motivo, por ejemplo, porque el usuario ha cancelado el proceso de autenticación, entonces Giltza enviará una respuesta de error al proveedor de servicios, a través del navegador. La respuesta consiste en un mensaje *Response* como el descrito anteriormente, no firmado, sin ningún elemento *Assertion*, y con la siguiente información en el elemento *Status*:

- *StatusCode*: código de estado de primer nivel.
- *StatusCode.StatusCode*: código de estado subordinado (puede no estar presente).
- *StatusMessage*: descripción adicional del error. (puede no estar presente)

A continuación, se listan los valores de ambos códigos de estado y de *StatusMessage* (si lo hay) para algunos errores típicos, y su correspondiente significado:

Nombre	Descripción
Responder, AuthnFailed	El usuario ha cancelado explícitamente el proceso de autenticación.
Responder, AuthnFailed, RiskyAuthnContextException	Se ha denegado el acceso al usuario debido a que está accediendo bajo un contexto de riesgo.
Requester, NoAuthnContext, UnrecognizedAuthnRequirementsException	No se reconoce ninguno de los requisitos de autenticación especificados en el elemento <i>RequestedAuthnContext</i> .
Responder, (ausente), MissingMappedAttributeException	Alguno de los atributos definidos para el elemento <i>Subject.NameID</i> no está presente en el repositorio.

6.1.4.3. Ejemplo de respuesta de error

En este punto, se muestra un ejemplo de repuesta de error SAML 2.0:

```
<ns4:Response ID="response-0c8bd74bf0af3e5071e7843e97bc656b29b6bf05ccaa0cb5bc9b384f7c246106"
InResponseTo="lidlojljhokgckhmpggpkibfaapjeeefdhgce" Version="2.0" IssueInstant="2018-04-
23T08:01:05.795Z" Destination="http://localhost:8081/minimalist-oauth-rp/samlCallback"
xmlns:ns2="http://www.w3.org/2000/09/xmldsig#" xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:ns4="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:ns3="http://www.w3.org/2001/04/xmlenc#">
  <ns4:Status>
    <ns4:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Responder">
      <ns4:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:AuthnFailed"/>
    </ns4:StatusCode>
  </ns4:Status>
</ns4:Response>
```

Otro ejemplo de respuesta de error SAML 2.0 con una descripción adicional del error:

```
<ns4:Response ID="response-0c8bd74bf0af3e5071e7843e97bc656b29b6bf05ccaa0cb5bc9b384f7c246106"
InResponseTo="lidlojljhokgckhmmppgpkibfaapjeeefdhgce" Version="2.0" IssueInstant="2018-04-
23T08:01:05.795Z" Destination="http://localhost:8081/minimalist-oauth-rp/samlCallback"
xmlns:ns2="http://www.w3.org/2000/09/xmlsig#" xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:ns4="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:ns3="http://www.w3.org/2001/04/xmlenc#">
  <ns4:Status>
    <ns4:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Responder">
      <ns4:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:AuthnFailed"/>
    </ns4:StatusCode>
    <ns4:StatusMessage>
      RiskyAuthnContextException
    </ns4:StatusMessage>
  </ns4:Status>
</ns4:Response>
```

7. Integración con el servicio de firma

7.1. Resumen de las operaciones de firma disponibles en Giltza

En este punto, se detallan brevemente todas las posibles operaciones que dispone Giltza para efectuar firmas electrónicas avanzadas de documentos y las combinaciones que se pueden utilizar, dependiendo del mecanismo de firma utilizado.

NOTA 1: El uso del ensobrado *Internally Detached* sólo se recomienda para la realización de las firmas XMLDSig/XAdES en serie, es decir, para las firmas posteriores a la primera y que irán incluidas en el nodo *<CounterSignature>*. En caso de realizar la primera firma con esta modalidad de ensobrado, Zain no será capaz de validarla, ya que actualmente, no está soportada.

NOTA 2: Todas las operaciones de firma disponibles en Giltza soportan la inclusión de un sellado de tiempo.

7.1.1. Básicas

Operación firma	Giltza	Idazki < 3.0	Idazki >= 3.0
PDF/PAdES (EPES)	Disponible	Disponible	Disponible
XMLDSig/XAdES Enveloped (EPES)	Disponible	Disponible	Disponible
XMLDSig/XAdES Enveloping (EPES)	Disponible	No disponible	Disponible
XMLDSig/XAdES Enveloping con elemento Manifest (EPES)	Disponible	No disponible	Disponible
XMLDSig/XAdES Internally Detached (EPES)	Disponible	Disponible	Disponible
XMLDSig/XAdES Externally Detached (EPES)	Disponible	Disponible	Disponible
XMLDSig/XAdES Externally Detached a partir de hash (EPES)	Disponible	Disponible	Disponible
CMS/CAAdES Attached (EPES)	Disponible	No disponible	Disponible
CMS/CAAdES Detached (EPES)	Disponible	No disponible	Disponible

7.1.2. Por lotes

Operación firma	Giltza	Idazki < 3.0	Idazki >= 3.0
PDF/PAdES (EPES)	Disponibile	Disponibile	Disponibile
PDF/PAdES (EPES) con diferente apariencia en la firma visible	Disponibile	Disponibile	Disponibile
XMLDSig/XAdES Enveloped (EPES)	Disponibile	No disponible	Disponibile
XMLDSig/XAdES Enveloping (EPES)	Disponibile	No disponible	Disponibile
XMLDSig/XAdES Enveloping con elemento Manifest (EPES)	Disponibile	No disponible	Disponibile
XMLDSig/XAdES Internally Detached (EPES)	Disponibile	Disponibile	Disponibile
XMLDSig/XAdES Externally Detached (EPES)	Disponibile	Disponibile	Disponibile
XMLDSig/XAdES Externally Detached a partir de hash (EPES)	Disponibile	Disponibile	Disponibile
CMS/CAAdES Attached (EPES)	Disponibile	No disponible	Disponibile
CMS/CAAdES Detached (EPES)	Disponibile	No disponible	Disponibile
Combinación de diferentes perfiles de firma (PAdES + XAdES + CAAdES)	Disponibile	No disponible	Disponibile

7.1.3. Múltiples

Operación firma	Tipo	Giltza	Idazki < 3.0	Idazki >= 3.0
PDF/PAdES (EPES)	Serie (countersign)	Disponibile	Disponibile	Disponibile
XMLDSig/XAdES Externally Detached (EPES)	Paralelo (cosign)	Disponibile	Disponibile	Disponibile
XMLDSig/XAdES Externally Detached (EPES)	Serie (countersign)	Disponibile	Disponibile	Disponibile

7.2. Kit de integración con el servicio de firma

Este kit de integración del servicio de firma describe los elementos necesarios para integrar una aplicación web o una aplicación móvil nativa con los servicios avanzados de firma proporcionados por el sistema Giltza.

NOTA: Para usar el servicio de firma, es necesario disponer de un token OAuth 2.0 que habilite el acceso a las identidades de firma. A tal fin, la aplicación debe registrarse como Proveedor de Servicios (RP) en Giltza y disponer de las credenciales para su aplicación emitidas por **izenpe**, en particular el *api_key*. Si no es así, se debe solicitar a **izenpe** dicha información (ver *Registro de una aplicación*).

Se trata de un servicio de firma en el que el servidor orquesta el proceso de firmas electrónicas avanzadas de documentos (firmas PAdES y XAdES que, opcionalmente, incluyen un sello de tiempo).

Del mismo modo, se puede solicitar la firma electrónica avanzada partiendo del hash de un documento, así como por lotes.

Dependiendo de si el proceso de firma electrónica avanzada se realiza para un sólo documento/ partiendo de un hash o por lotes, el proceso completo varía ligeramente:

Proceso de firma electrónica avanzada con documento único/ partiendo de un hash

El proceso completo se realiza en seis pasos (uno de ellos opcional), a saber:

1. Obtención del token OAuth2.0
2. Creación del proceso de firma
3. Ejecución del proceso de firma
4. Obtención de la información de un proceso de firma de documentos (opcional)
5. Obtención del documento firmado
6. Eliminación del proceso de firma

Cada proceso de firma contra Giltza necesita el intercambio de cinco peticiones de servicio, siendo las que se muestran a continuación:

1. Obtención token OAuth 2.0. El endpoint del servicio es:

<https://eid.izenpe.com/trustedx-authserver/oauth/esignsp/token>

2. Creación del proceso de firma. El endpoint del servicio es el siguiente:

https://eid.izenpe.com/trustedx-resources/esignsp/v2/signer_processes

3. Ejecución del proceso de firma, cuyo endpoint es:

<https://eid.izenpe.com/trustedx-resources/esignsp/v2/ui>

4. Obtención de la información de un proceso de firma de documentos (opcional), cuyo endpoint es:

https://eid.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}

5. Obtención del documento firmado. El endpoint del servicio es:

https://eid.izenpe.com/trustedx-resources/esignsp/v2/documents/<document_id>/content

6. Eliminación del proceso de firma. El endpoint del servicio es el siguiente:

https://eid.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/<document_id>

Además, el proceso de firma actualmente ofrece la opción de firmar mediante Idazki Desktop.

Proceso de firma electrónica avanzada mediante lotes

El proceso completo se realiza en ocho pasos (uno de ellos opcional), a saber:

1. Obtención del token OAuth2.0
2. Creación de un recurso
3. Creación del proceso de firma
4. Ejecución del proceso de firma
5. Obtención de la información de un proceso de firma de documentos (opcional)
6. Obtención de la información de los documentos firmados
7. Eliminación de un recurso
8. Eliminación del proceso de firma

Cada proceso de firma contra Giltza necesita el intercambio de siete peticiones de servicio, siendo las que se muestran a continuación:

1. Obtención token OAuth 2.0. El *endpoint* del servicio es:

<https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/token>

2. Creación de un recurso. La URL es la siguiente:

<https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents>

3. Creación del proceso de firma. El *endpoint* del servicio es el siguiente:

https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes

4. Ejecución del proceso de firma, cuyo *endpoint* es:

<https://eidas.izenpe.com/trustedx-resources/esignsp/v2/ui>

5. Obtención de la información de un proceso de firma de documentos (opcional), cuyo *endpoint* es:

https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}

6. Obtención de la información de los documentos firmados. El *endpoint* del servicio es:

https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}/documents

7. Eliminación de un recurso, cuyo *endpoint* es:

https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/<document_id>

8. Eliminación del proceso de firma. El *endpoint* del servicio es el siguiente:

https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/<signature_process_id>

Además, el proceso de firma actualmente ofrece la opción de firmar mediante Idazki Desktop.

NOTA: Si el usuario ya ha sido autenticado previamente antes de iniciar el proceso de firma se hará SSO y no volverá a autenticarse. Por el contrario, si el usuario no está autenticado antes de iniciar el proceso de firma, entonces el mismo proceso iniciará de forma automática una autenticación, excepto si se solicita firmar con Idazki Desktop.

7.2.1. Obtención del token OAuth 2.0

El token de acceso se obtiene mediante la petición y la respuesta descritas a continuación.

7.2.1.1. Petición

Para solicitar el token de acceso, la aplicación debe mandar una petición POST con el siguiente contenido:

```
POST /trustedx-authserver/oauth/esignsp/token HTTP/1.1
Host: eidas.izenpe.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Basic <api_key>

grant_type=client_credentials&
scope=urn:safelayer:eidas:sign:process:document
```

Los parámetros de la petición son:

Nombre	Valor	Presencia	Descripción
api_key	<api_key>	Obligatorio	API Key obtenido en el registro de la aplicación
grant_type	client_credentials	Obligatorio	Valor definido por OAuth 2.0 para obtener un token de acceso
scope	urn:safelayer:eidas:sign:process:document	Obligatorio	Scope exigido para iniciar el proceso de firma de documentos

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
```

7.2.1.2. Respuesta

```
"expires_in":600,  
"token_type":"Bearer",  
"access_token":"<access_token>"  
}
```

Los parámetros de la respuesta de ejemplo son:

- *expires_in*: segundos durante los que el token de acceso es válido.
- *token_type*: tipo de token de acceso. Siempre tendrá el valor *Bearer*.
- *access_token*: token que la aplicación deberá incluir en peticiones a los servicios en nombre del usuario autenticado.

7.2.2. Creación del proceso de firma

A la hora de crear un proceso de firma, Giltza diferencia entre las siguientes opciones:

- *Firma de un documento*: se pueden crear procesos de firma para firmar documentos PDF (firmas PAdES-Basic, PAdES-BES, PAdES-EPES y PAdES-T) y cualquier otro formato de documento (firmas XMLDSig, XAdES-BES, XAdES-EPES, XAdES-T, CMS, CAdES-BES, CAdES-EPES y CAdES-T).
- *Firma de varios objetos de datos a partir de sus hashes*: sólo se pueden realizar firmas en formato XMLDSig/XAdES ExternallyDetached.
- *Firma de varios documentos (lotes)*: se pueden crear procesos de firma para firmar documentos PDF (firmas PAdES-Basic, PAdES-BES, PAdES-EPES y PAdES-T) y cualquier otro formato de documento (firmas XMLDSig, XAdES-BES, XAdES-EPES, XAdES-T, CMS, CAdES-BES, CAdES-EPES y CAdES-T).
 - Se permite la firma de varios documentos PDF, cada uno de ellos, con una apariencia visible de firma completamente diferente.
 - Si el proceso de firma se crea a través de Bak, BakQ o Bak + OTP, se puede solicitar la firma de varios documentos con diferentes perfiles de firma (PAdES yXAdES).
 - Si se tiene instalado Idazki Desktop v3.0, también será posible realizar este tipo de firma mediante un certificado en tarjeta (PKCS#11) o en software (PKCS#12).
- *Multifirma*: se pueden crear procesos de firmas múltiples para documentos PDF (paralelo) o cualquier otro formato de documento (paralelo y serie).

NOTA: En el primer y el tercer caso, la operación también crea el recurso que corresponde al proceso de firma creado. Si el proceso se ejecuta con éxito, los correspondientes documentos firmados o firmas deberán obtenerse mediante la operación *Obtención del documento firmado*.

A estos procesos de firma se les puede incluir un sello de tiempo, de manera que se incorpora a la firma la capacidad de no repudio.

El proceso de firma se obtiene mediante la petición y la respuesta descritas a continuación.

7.2.2.1. Petición

<Content-Type>

Existen tres posibilidades, dependiendo de:

- Si la operación se ejecuta para firmar un documento que se proporciona

```
Content-Type: multipart/form-data;boundary=<part_boundary_mark>
```

- Si se ejecuta para firmar varios objetos de datos externos a partir de sus hashes

```
Content-Type: application/json
```

- Si la operación se ejecuta para firmar los documentos de un conjunto de recursos

```
Content-Type: application/json
```

<Cuerpo>

Al igual que en el apartado anterior, existen tres posibilidades, dependiendo de si la operación se ejecuta para:

Firmar un documento que se proporciona

El cuerpo contiene una entidad *multipart/form-data* con dos partes: una de ellas ("*process*") contiene un objeto JSON con los datos necesarios para crear el proceso de firma; la otra ("*document*") contiene el documento que se quiere firmar. La cabecera *Content-Type* de esta parte indica el tipo de este documento (un PDF si el valor de la cabecera es *application/pdf*, un documento XML si el valor es *text/xml; charset="UTF-8"*, etc.). Por otro lado, la cabecera *Content-Transfer-Encoding* de la parte "*document*" solo es necesaria cuando el documento que se quiere firmar está codificado en binario (un documento PDF, una imagen PNG, etc.)

Se realiza mediante una petición HTTP POST de la siguiente forma:

```

POST /trustedx-resources/esignsp/v2/signer_processes HTTP/1.1
Authorization: Bearer <access_token>
Host: eidas.izenpe.com
Content-Type: multipart/form-data;boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

--WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="process"
Content-Type: application/json

{
  "process_type" : {string},
  "signer" : {
    ...
  },
  "labels": [
    [<labels>]
  ],
  "ui_locales" : [
    [<ui_locales>]
  ],
  "finish_callback_url" : "<callback_url>",
  "views" : {
    ...
  },
  "timestamp" : {
    "provider_id" :{string}
  }
}
--WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="document";
Content-Type: application/pdf
Content-Transfer-Encoding: binary

<document>
--WebKitFormBoundary7MA4YWxkTrZu0gW

```

Parámetros:

Nombre	Valor	Presencia	Descripción
access_token	<access_token>	Obligatorio	Token obtenido en el mensaje anterior
process	Objeto JSON	Obligatorio	Información sobre el proceso de firma
document	Contenido	Obligatorio	Documento para firmar

Firmar varios objetos de datos externos a partir de sus hashes

En este caso, el cuerpo contiene un objeto JSON con los datos necesarios para crear el proceso de firma.

Se realiza mediante una petición HTTP POST de la siguiente forma:

```
POST /trustedx-esignsp/v2/signer_processes HTTP/1.1
Authorization: Bearer <access_token>
Host: eidas.izenpe.com
Content-Type: application/json

{
  "process_type" : {string},
  "signer" : {
    ...
  },
  "labels": [
    [<labels>]
  ],
  "ui_locales" : [
    [<ui_locales>]
  ],
  "finish_callback_url" : "<callback_url>",
  "views" : {
    ...
  },
  "timestamp" : {
    "provider_id" :{string}
  }
}
```

Firmar los documentos de un conjunto de recursos

En este caso, el cuerpo contiene un objeto JSON con los datos necesarios para crear el proceso de firma. La propiedad *documents* es obligatoria y la propiedad *signer* es opcional.

Se realiza mediante una petición HTTP POST de la siguiente forma:

```
POST /trustedx-esignsp/v2/signer_processes HTTP/1.1
Authorization: Bearer <access_token>
Host: eidas.izenpe.com
Content-Type: application/json

{
  "process_type" : {string},
  "signer" : {
    ...
  },
  "labels": [
    [<labels>]
  ],
  "documents" : [{
    "id" : {string}
  }]
  "ui_locales" : [
    [<ui_locales>]
  ],
  "finish_callback_url" : "<callback_url>",
  "views" : {
    ...
  },
  "timestamp" : {
    "provider_id" :{string}
  }
}
```

Los parámetros descritos son:

Nombre	Valor	Presencia	Descripción
process_type	<ul style="list-style-type: none"> <i>urn:safelayer:eidaz:processes:document:sign</i>: el proceso utiliza una identidad de firma gestionada por Giltza o por una aplicación externa. Al principio del proceso, no se autenticará al usuario ni se obtendrá su autorización para consultar las identidades de firma con las que se cuenta, si no que, ambas cosas se realizarán más adelante, en el caso de que el usuario seleccione un esquema de identidades de firma de tipo eSigP <i>urn:safelayer:eidaz:processes:document:sign:esigp</i>: el proceso utiliza una identidad de firma gestionada por Giltza. Al principio del proceso, se autenticará al usuario y se obtendrá su autorización para consultar las identidades de firma con las que cuenta 	Obligatorio	Definición del proceso de firma
signer	Objeto JSON	Obligatorio	Definición de la firma
labels	String	Opcional	Lista de etiquetas
ui_locales	String	Opcional	Lista de idiomas
finish_callback_url	String	Obligatorio	URL de <i>callback</i>
views	Objeto JSON	Opcional	Definición de las <i>views</i> (no aplica a Idazki Desktop)
timestamp	String	Opcional	Especifica la política (<i>provider_id</i>) que se debe usar para el sellado de tiempo. Si no se especifica, la política se seleccionará por la plataforma.

NOTA: Todos los objetos JSON mencionados en la columna Valor de la tabla correspondiente a los parámetros se describen detalladamente en *Configuración de características de la firma*. Para ver información adicional en la pantalla de confirmación de firma del documento, emplear el parámetro *views* tal y como se describe en el capítulo mencionado.

<labels>

El valor de las etiquetas del parámetro **labels** depende del alcance de la aplicación. Estas etiquetas actúan como filtros que deben aplicarse sobre el conjunto de identidades de firma del usuario con vistas a seleccionar la que deberá utilizarse para firmar el documento.

Las distintas combinaciones posibles se muestran en la siguiente tabla:

RP	Etiquetas	Ejemplo JSON
Para usar clave Bak	bak, serverid	"labels": [{"bak", "serverid"}]
Para usar clave BakQ	bakq, serverid	"labels": [{"bakq", "serverid"}]
Para usar Idazki Desktop	idazki	"labels": [{"idazki"}]
Para usar Idazki Desktop con comprobación de TSL	idazki_tsl	"labels": [{"idazki_tsl"}]
Para usar clave BakQ + Idazki Desktop	idazki_bakq	"labels": [{"idazki_bakq"}]
Para usar clave BakQ + Idazki Desktop con comprobación de TSL	idazki_bakq_tsl	"labels": [{"idazki_bakq_tsl"}]
Para usar clave Profesional + Idazki Desktop	idazki_profesional	"labels": [{"idazki_profesional"}]
Para usar clave funcionario + Idazki Desktop	idazki_pep	"labels": [{"idazki_pep"}]
Para usar clave Profesional + Idazki Desktop con comprobación de TSL	idazki_profesional_tsl	"labels": [{"idazki_profesional_tsl"}]
Para usar clave funcionario + Idazki Desktop con comprobación de TSL	idazki_pep_tsl	"labels": [{"idazki_pep_tsl"}]
Para usar clave Profesional	profesional	"labels": [{"profesional"}]
Para usar clave funcionario	funcionario	"labels": [{"funcionario"}]
Para usar cualquier identidad de firma disponible en Giltza	izenpe	"labels": [{"izenpe"}]
Para usar cualquier identidad de firma disponible en Giltza con comprobación de TSL	izenpe_tsl	"labels": [{"izenpe_tsl"}]
Para usar, de forma dinámica, la clave asociada al valor del CIF incluido en el certificado que se ha usado en la autenticación. Sólo para Giltza Profesional.	CIF * etiqueta dinámica	"labels": [{"CIF:<valor_cif>}]

7.2.2.2. Respuesta

Si todo es correcto, la respuesta HTTP tendrá la siguiente estructura:

```

HTTP/1.1 201 Created
Location: https://eid.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/<signature_process_id>
Content-Type: application/json

{
  "process_type" : "urn:safelayer:eid:processes:document:sign",
  "id" : <signature_id>,
  "self" : "Location:https://eid.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/<signature_process_id>",
  "tasks" : {
    "pending" : [
      {
        "type" : "UserBrowserTask",
        "id" : "<id>",
        "url" : "<redirection_url>"
      }
    ]
  },
  "documents" : [
    {
      "url" : "<document_url>"
    }
  ]
}

```

Los parámetros de la respuesta son:

Nombre	Valor	Descripción
signature_process_id	<signature_process_id>	Identificador del proceso de firma
process_type	urn:safelayer:eid:processes:document:sign urn:safelayer:eid:processes:document:sign:esigp	Tipo de proceso de firma
id	<signature_id>	Identificador de la firma
redirection_url	https://eid.izenpe.com/trustedx-resources/esignsp/v2/ui?signerProcessId=<signature_id>	URL de redirección para el siguiente mensaje
document_url	https://eid.izenpe.com/trustedx-resources/esignsp/v2/documents/<document_id>	URL para obtener el documento firmado

7.2.3. Creación de un recurso

Este recurso puede crearse con dos finalidades:

- Para realizar varias firmas del documento que se proporciona: actualmente, solo se pueden crear recursos de este tipo que contengan un documento y una definición de firma PAdES-Basic, PAdES-BES, PAdES-EPES, PAdES-T, XMLDSig, XAdES-BES, XAdES-EPES, XAdES-T, CMS, CAdES-BES, CAdES-EPES o CAdES-T.

- Para realizar la firma de varios objetos de datos externos a partir de sus respectivos hashes: actualmente, solo se pueden crear recursos de este tipo que contengan un documento vacío y una definición de firma XMLDSig/XAdES Externally Detached.

La operación resulta útil en los siguientes escenarios:

- Si se quiere firmar un lote de documentos: para subir los documentos y opcionalmente las definiciones de firmas, para después crear el correspondiente proceso de firma.
- Si se quiere firmar un documento y delegar en Giltza la orquestación del proceso del firma: para subir el documento y opcionalmente las definiciones de firma, para después crear el correspondiente proceso de firma.
- Si se quiere firmar un documento y orquestar también el proceso de firma.

El proceso de firma se obtiene mediante la petición y la respuesta descritas a continuación.

7.2.3.1. Petición

<Content-Type>

Existen tres posibilidades, dependiendo de:

- Si el recurso se crea para firmar un documento que se proporciona junto a las definiciones de firmas.

```
Content-Type: multipart/form-data;boundary=<part_boundary_mark>
```

- Si el recurso se crea para firmar varios objetos de datos externos a partir de sus hashes.

```
Content-Type: application/json
```

- Si el recurso se crea para firmar un documento que se proporciona sin definiciones de firmas: en este caso el *Content-Type* será el que corresponda al tipo de documento que se quiera firmar (*application/pdf*, *text/xml*, etc.).

<Cuerpo>

Al igual que en el apartado anterior, existen tres posibilidades, dependiendo de si la operación se ejecuta para:

El recurso se crea para firmar un documento que se proporciona junto a las definiciones de firmas

Contiene una entidad *multipart/form-data* con dos partes: una de ellas ("*signers*") es opcional y contiene un array JSON con los datos necesarios para crear las definiciones de firmas del recurso. En el caso de que no se proporcionen datos de ninguna definición de firma (i.e. que la parte "*signers*" contenga un *array* vacío), éstas deberán proporcionarse posteriormente al recurso por medio de la operación Crear una definición de firma. La otra parte ("*document*") contiene el documento que se quiere asignar al recurso (que a la postre será el documento que se firme). La cabecera *Content-Type* de esta parte indica el tipo de este documento (un PDF si el valor de la cabecera es *application/pdf*, un documento XML si el valor es *text/xml; charset="UTF-8"*, etc). Por otro lado, la cabecera *Content-*

Transfer-Encoding de la parte "document" solo es necesaria cuando el documento que se quiere firmar está codificado en binario (un documento PDF, una imagen PNG, etc.)

```
POST /trustedx-resources/esignsp/v2/documents HTTP/1.1
Authorization: Bearer <access_token>
Host: eidas.izenpe.com
Content-Type: multipart/form-data;boundary= ----WebKitFormBoundary7MA4YWxkTrZu0gW

-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="signers"
Content-Type: application/json
[
  {
    "signature_policy_id" : {string},
    "parameters" : {Parameters}
  }
]
-----WebKitFormBoundary7MA4YWxkTrZu0gW
```

Se realiza mediante una petición HTTP POST de la siguiente forma:

```
Content-Disposition: form-data; name="document";
Content-Type: application/pdf
Content-Transfer-Encoding: binary

<doc>
- ----WebKitFormBoundary7MA4YWxkTrZu0gW--
```

El recurso se crea para firmar un documento que se proporciona sin definiciones de firmas

En este caso, el cuerpo contiene el documento que se quiere firmar.

El recurso se crea para firmar varios objetos de datos externos a partir de sus hashes

El cuerpo contiene un objeto JSON con los datos necesarios para crear el proceso de firma.

```
{
  "signature_policy_id" : {string},
  "parameters" : {Parameters}
}
```

NOTA: Ver el apartado *Configuración de características de la firma* para más información acerca de los parámetros JSON arriba mencionados.

7.2.3.2. Respuesta

Si todo es correcto, la respuesta HTTP tendrá la siguiente estructura:

```
HTTP/1.1 201 Created
Location: https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/{doc_and_sigdefs_id}
Content-Type: application/json

{
  "id" : <id>,
  "self" : "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/{doc_and_sigdefs_id}",
  "signers" : [
    {
      "id" : "<signer_id>"
      "self" : "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/{doc_and_sigdefs_id}/signers/<signer_id>"
    }
  ]
}
```

Los parámetros de la respuesta son:

Propiedad	Descripción
signature_process_id	Identificador del proceso de firma
process_type	Tipo de proceso de firma
id	Identificador de la firma
redirection_url	URL de redirección para el siguiente mensaje
document_url	URL para obtener el documento firmado

7.2.4. Ejecución del proceso de firma

La ejecución del proceso de firma se realiza mediante la petición y respuesta descritas a continuación.

7.2.4.1. Petición

La ejecución del proceso de firma de un documento se ordena enviando una respuesta de redirección al navegador con el que el usuario haya solicitado la firma. La URL a la que se tiene que redirigir al navegador debe extraerse de la respuesta obtenida en la operación de *Creación del proceso de firma* (<redirection_url>).

El mensaje será, por tanto, similar al siguiente:

```
HTTP/1.1 302 Found
Location:<redirection_url>
```

Donde *redirection_url* tendrá un parámetro *<signature_id>*, que coincide con el id obtenido anteriormente.

El navegador reaccionará enviando la siguiente petición HTTP a Giltza.

```
GET /trustedx-resources/esignsp/v2/ui?signerProcessId=<signature_id>
Host: eidas.izenpe.com
```

Tras la redirección, se solicitará la autenticación del usuario si no tenía aún una sesión iniciada en Giltza. Si hay varias identidades de firma para las etiquetas pedidas en el paso anterior, aparecerá por pantalla un selector para que el usuario escoja la identidad que quiere usar. El uso de todas las etiquetas descritas en el paso anterior para cada caso evitará la aparición de dicha selección.

7.2.4.2. Respuesta

La respuesta es un mensaje HTTP GET en la URL indicada en el parámetro *<finish_callback_url>* de la petición generada en Creación del proceso de firma.

```
GET <finish_callback_url_path>?status=finished
Host: <finish_callback_url_host>
```

Los parámetros son:

Nombre	Valor	Descripción
finish_callback_url_pat	<finish_callback_url_path>	Path incluido en la URL
finish_callback_url_hos t	<finish_callback_url_host>	Nombre de host y puerto incluidos en la URL
status	finished / failed / canceled	Indica el resultado del proceso mediante uno de los siguientes valores: <ul style="list-style-type: none"> <i>finished</i>: el proceso ha terminado y el documento ha sido firmado con éxito. <i>failed</i>: el proceso ha terminado. No obstante, el documento no ha sido firmado porque se ha producido un error. <i>canceled</i>: el proceso ha terminado mediante la cancelación de su ejecución. En consecuencia, el documento no ha sido firmado.

7.2.5. Obtención de la información de un proceso de firma de documentos (opcional)

La obtención de la información de un proceso de firma de documentos se realiza mediante la petición y la respuesta que se describen a continuación.

7.2.5.1. Petición

La petición siguiente obtiene información correspondiente al proceso de firma de documentos solicitado a Giltza mediante una petición HTTP GET.

```
GET /trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}
Authorization: Bearer <token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *signature_process_id*: identificador del proceso de firma de documentos del que se quiere obtener información (obligatorio).

7.2.5.2. Respuesta

Si todo es correcto, la respuesta HTTP tendrá la siguiente estructura:

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id" : {string},
  "self" : {string},
  "process_type" : {string},
  "signing_information" : {
    "labels" : [ {string} ],
    "sign_identity" : {
      "id" : {string},
      "url" : {string}
    }
  },
  "result" : {
    "status": {string},
    "details": {
      "message": {string}
    }
  },
  "tasks" : {
    "pending" : [ {
      "id" : {string},
      "type" : {string},
      "url" : {string}
    } ]
  },
  "documents" : [ {
    "id" : {string},
    "url" : {string},
    "content" : {string},
    "result" : {
      "status": {string},
      "details": {
        "message": {string}
      }
    }
  } ]
}

```

Los parámetros de la respuesta son:

Propiedad	Descripción
id	Identificador del proceso de firma. Esta propiedad es opcional y sólo está presente cuando la representación corresponde a un proceso de firma que ya ha sido creado.
self	URL del proceso de firma. Esta propiedad es opcional y sólo está presente cuando la representación corresponde a un proceso de firma que ya ha sido creado.

process_type	<p>Tipo de proceso de firma. Esta propiedad puede tener dos valores:</p> <ul style="list-style-type: none"> • <i>urn:safelayer:eidas:processes:document:sign:esigp</i>: el proceso utiliza una identidad de firma gestionada por la plataforma Giltza. Al principio del proceso se autentificará al usuario y se obtendrá su autorización para consultar las identidades de firma con las que cuenta. • <i>urn:safelayer:eidas:processes:document:sign</i>: el proceso utiliza una identidad de firma gestionada por Giltza o por una aplicación externa. Al principio del proceso, no se autentificará al usuario ni se obtendrá su autorización para consultar las identidades de firma con las que cuenta, sino que ambas cosas se realizarán más adelante, en el caso de que el usuario seleccione un esquema de identidades de firma de tipo eSigP.
signing_information	<p>Información sobre la identidad de firma que utiliza el proceso. Esta propiedad es opcional y solo está presente si el usuario ha seleccionado ya la identidad de firma con la que quiere firmar (en el caso de que se trate de una identidad de firma gestionada por Giltza, es decir, de una identidad perteneciente a un esquema del proveedor de firma) o bien ha indicado que quiere firmar con una identidad de firma gestionada por una aplicación externa, es decir, con una identidad perteneciente a un esquema de aplicación externa.</p>
signing_information.labels	<p>Etiquetas de la identidad de firma seleccionada por el usuario, en el caso de que ésta sea una identidad de firma gestionada por Giltza. Alternativamente, etiquetas del esquema seleccionado por el usuario, el caso de que éste sea un esquema de aplicación externa.</p>
signing_information.sign_identity	<p>Esta propiedad es opcional y solo está presente si el usuario ha seleccionado una identidad de firma gestionada por Giltza.</p>
signing_information.sign_identity.id	<p>Identificador de la identidad de firma seleccionada por el usuario y gestionada por Giltza.</p>
signing_information.sign_identity.url	<p>URL de acceso a la identidad de firma seleccionada por el usuario y gestionada por Giltza.</p>
result	<p>Resultado del proceso de firma. Esta propiedad sólo existe en los procesos de firma que hayan finalizado.</p>
result.status	<p>Estado final del proceso de firma:</p> <ul style="list-style-type: none"> • <i>finished</i>: el proceso ha terminado después de que todos los documentos hayan sido firmados con éxito. • <i>failed</i>: el proceso ha terminado sin que se haya firmado ningún documento porque se ha producido un error. Este estado se comprobará para detectar si el usuario ha solicitado la actualización de Idazki Desktop. • <i>failed_documents</i>: el proceso ha terminado y se han firmado algunos documentos, pero no todos. • <i>canceled</i>: el proceso ha terminado sin que se haya firmado ningún documento porque su ejecución ha sido cancelada.

result.details.message	<p>Esta propiedad sólo está presente si se detallan las condiciones en las que ha acabado del proceso de firma y su valor es el mensaje con esta información. Habitualmente se emplea para que un mecanismo de firma externo proporcione detalles de por qué un proceso ha finalizado con error (estado <i>failed</i>).</p> <p>NOTA: en caso de necesitar averiguar si un usuario ha iniciado el proceso de actualización de Idazki Desktop, habrá que comprobar que el valor de este atributo es igual a <i>ERR_UPDATED</i>.</p>
tasks	Información que proporciona Giltza de las tareas de las que consta el proceso. Actualmente, Giltza informa solo de tareas pending, es decir, de tareas que tiene que realizar la aplicación cliente para que el proceso continúe.
tasks.pending[]	Información sobre las tareas que tiene que realizar la aplicación cliente para que el proceso de firma continúe. Actualmente, Giltza indica siempre una sola tarea que, además, es siempre de tipo <i>UserBrowserTask</i> .
tasks.pending[].id	Identificador de una de las tareas que tiene que realizar la aplicación cliente.
tasks.pending[].type	Tipo de una de las tareas que tiene que realizar la aplicación cliente. Actualmente esta propiedad tiene siempre el valor <i>UserBrowserTask</i> . Este valor identifica un tipo de tarea que consiste en redirigir el navegador del usuario a una URL determinada.
tasks.pending[].url	Esta propiedad solo está presente para una tarea pending que sea de tipo <i>UserBrowserTask</i> y su valor es la URL a la que la aplicación cliente tiene que redirigir el navegador del usuario para que dicha tarea se tenga por realizada.
documents[]	Información de los recursos que están asociados al proceso de firma.
documents[].url	URL de uno de los recursos.
documents[].id	Identificador de uno de los recursos.
documents[].content	URL del documento que contiene uno de los recursos.
documents[].result	<p>Resultado del proceso de firma para el documento que contiene uno de los recursos:</p> <ul style="list-style-type: none"> • <i>finished</i>: el proceso ha terminado y el documento ha sido firmado con éxito • <i>failed</i>: el proceso ha terminado y el documento no ha sido firmado porque se ha producido un error • <i>canceled</i>: el proceso ha terminado al ser cancelada su ejecución y el documento no ha sido firmado.
documents[].details.message	Mensaje que describe el error cuando el resultado del proceso de firma para el documento que contiene uno de los recursos es <i>failed</i> .

7.2.6. Obtención de la información de los documentos firmados

La obtención de la información de los documentos firmados en un proceso de firma, en particular, del resultado obtenido al realizar cada una de las firmas, se realiza mediante la petición y la respuesta que se describen a continuación.

7.2.6.1. Petición

La petición siguiente obtiene información correspondiente a cada una de las firmas solicitadas a Giltza mediante una petición HTTP GET.

```
GET /trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}/documents
Authorization: Bearer <token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *signature_process_id*: identificador del proceso de firma de documentos con el que se han firmado los documentos (obligatorio).

7.2.6.2. Respuesta

Si todo es correcto, la respuesta HTTP tendrá la siguiente estructura:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "documents" : [
    {
      "id" : {string},
      "url" : {string},
      "content" : {string},
      "result" : {
        "status" : {string},
        "details" : {
          "message" : {string}
        }
      }
    }
  ]
}
```

Los parámetros de la respuesta son:

Propiedad	Descripción
documents	Información de los documentos firmados en el proceso
documents.id	Identificador de uno de los documentos
documents.url	URL del recurso que corresponde a uno de los documentos

documents.content	URL de uno de los documentos
documents.result	<p>Resultado de la firma de uno de los documentos:</p> <ul style="list-style-type: none"> • <i>finished</i>: el documento ha sido firmado con éxito. • <i>failed</i>: el documento no ha sido firmado porque se ha producido un error. • <i>anceled</i>: el documento no ha sido firmado porque el proceso de firma fue cancelado.
documents.details.message	Mensaje que describe el error en la firma de uno de los documentos. Esta propiedad solo está presente cuando el resultado de la firma del documento es <i>failed</i> .

7.2.7. Obtención del documento firmado

La obtención del documento firmado se realiza mediante la petición y la respuesta descritas a continuación.

7.2.7.1. Petición

Tras recibir el mensaje anterior correspondiente a la Ejecución del proceso de firma, el Proveedor de Servicios debe solicitar el documento firmado enviando el siguiente mensaje HTTP a Giltza.

```
GET /trustedx-resources/esignsp/v2/documents/<document_id>/content
Authorization: Bearer <access_token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *document_id*: identificador del documento firmado. Se obtiene de la respuesta descrita en Creación del proceso de firma (obligatorio).
- *access_token*: token de acceso obtenido en Obtención del token de acceso (obligatorio).

7.2.7.2. Respuesta

La respuesta obtenida será:

```
HTTP/1.1 200 OK
Content-Type: application/pdf

{documentSigned}
```

Donde *documentSigned* es el documento firmado en binario.

7.2.8. Eliminación de un recurso

La eliminación de un recurso se realiza mediante la petición y la respuesta descritas a continuación.

7.2.8.1. Petición

La petición siguiente elimina un recurso mediante HTTP DELETE.

```
DELETE /trustedx-resources/esignsp/v2/documents/{doc_and_sigdefs_id}
Authorization: Bearer <token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *doc_and_sigdefs_id*: Identificador del recurso que se quiere eliminar (obligatorio).

7.2.8.2. Respuesta

Si el recurso ha sido eliminado con éxito la respuesta no tiene cuerpo

```
HTTP/1.1 204 No Content
```

Para todos los demás casos, la respuesta tendrá un cuerpo y contendrá un objeto JSON que describa el error, por ejemplo:

```
HTTP/1.1 404 Not Found
Content-Type: application/json
...
```

7.2.9. Eliminación del proceso de firma

La eliminación de un proceso de firma se realiza mediante la petición y la respuesta descritas a continuación.

7.2.9.1. Petición

La petición siguiente elimina un proceso de firma de un documento mediante HTTP DELETE, además de eliminar el recurso correspondiente a dicho proceso.

```
DELETE /trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}
Authorization: Bearer <token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *signature_process_id*: identificador del proceso de firma del documento que se quiere eliminar (obligatorio).

7.2.9.2. Respuesta

Giltza elimina el proceso de firma de documento y responde con el siguiente mensaje:

```
HTTP/1.1 204 No content
```

7.2.10. Obtención del resultado de un proceso de firma

La obtención del resultado de un proceso de firma se realiza mediante la petición y la respuesta descritas a continuación.

7.2.10.1. Petición

La petición siguiente recupera el resultado de un proceso de firma mediante HTTP GET.

```
GET /trustedx-resources/esignsp/v2/signer_processes/{signature_process_id}/result
Authorization: Bearer <token>
Host: eidas.izenpe.com
```

Los parámetros de la petición son:

- *signature_process_id*: identificador del proceso de firma cuyo resultado se quiere obtener (obligatorio).

7.2.10.2. Respuesta

Si el resultado del proceso de firma se obtiene con éxito, la respuesta HTTP devuelta por Giltza contendrá con el siguiente mensaje:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "status": {string},
  "details": {
    "message": {string}
  }
}
```

Los parámetros de la respuesta son:

Propiedad	Presencia	Descripción
-----------	-----------	-------------

status	Obligatorio	Indica el resultado del proceso de firma de documentos: <ul style="list-style-type: none"> <i>finished</i>: el proceso ha terminado después de que todos los documentos hayan sido firmados con éxito <i>failed</i>: el proceso ha terminado sin que se haya firmado ningún documento porque se ha producido un error <i>failed_documents</i>: el proceso ha terminado y se han firmado algunos documentos, pero no todos <i>anceled</i>: el proceso ha terminado sin que se haya firmado ningún documento porque su ejecución ha sido cancelada
details.message	Opcional	Información adicional sobre el resultado del proceso de firma de documentos

7.3. Configuración de características de la firma

El objeto JSON correspondiente al parámetro *process* de la petición descrita en *Creación del proceso de firma* puede contener los siguientes elementos:

7.3.1. Signer

Contiene la definición de la firma del documento con la siguiente estructura:

```
{
  "signature_policy_id" : {string},
  "parameters" : {Parameters}
}
```

Sólo puede omitirse si el objeto JSON contiene la propiedad *documents[]* y cada uno de los recursos a los que apunta dicha propiedad contiene la definición de firma correspondiente.

El objeto JSON que contiene los datos a partir de los que se creará cada una de las definiciones de firma, tiene las siguientes propiedades:

Propiedad	Presencia	Descripción
signature_policy_id	Obligatorio	Si el recurso se crea para firmar un documento que se proporciona, la propiedad puede tener los siguientes valores: <ul style="list-style-type: none"> <i>urn:safelayer:idas:policias:sign:document:pdf</i>: definición de una firma PAdES de ETSI (ETSI TS 102778) <i>urn:safelayer:idas:policias:sign:document:xml</i>: definición de una firma W3C XMLDSig/XAdES (ETSI TS 101903) <i>urn:safelayer:idas:policias:sign:document:cms</i>: definición de una firma CMS/CADES (ETSI TS 101 733, ETSI TS 103 173, y ETSI EN 319 122)
parameters	Opcional	Valor que se asignará a la propiedad <i>parameters</i> a cada uno de los recursos que se crea. Esta propiedad puede omitirse en algunos elementos del <i>array</i>

7.3.3.1. Parameters

La propiedad *parameters* se amplía en formato JSON con distintos elementos según el formato de la firma.

7.3.3.1.1. FirmasPDF/PAdES

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type": {string},
  "certificate": {string},
  "default_digest_algorithm": {string},
  "estimated_signature_size": {number},
  "certification_level": {number},
  "contact_info": {string},
  "location": {string},
  "reason": {string},
  "signature_field": {
    "name": {string},
    "location": {
      "page": {
        "number": {string}
      },
      "rectangle": {
        "x": {number},
        "y": {number},
        "height": {number},
        "width": {number}
      }
    }
  },
  "appearance": {
    "background_image": {
      "binary": {string}
    },
    "foreground_image": {
      "binary": {string}
    },
    "signature_details": {signature_details}
  }
},
  "policy_identifier": {
    "policy_id": {
      "oid": {string}
    },
    "policy_hash": {
      "digest_algorithm_identifier": {
        "id": {string}
      },
      "digest_value": {string}
    },
    "policy_qualifiers": [{
      "type": {string},
      "uri": {string}
    }]
  },
  "commitments": [{
    "type": {string},
    "oid": {string}
  }]
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
-----------	--------	--------------	---------------	-------------

type	Obligatorio	Obligatorio	Obligatorio	<p>Tipo de firma ("pdf", "pades-bes", "pades-epes"):</p> <ul style="list-style-type: none"> • <i>pdf</i>: PAdES-Basic (ETSI TS 102778-2) • <i>pades-bes</i>: PAdES- BES (ETSI TS 102778-3) • <i>pades-epes</i>: PAdES- EPES (ETSI TS 102 778- 3)
certificate	Opcional	Ausente	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	Opcional	Opcional	Algoritmo de hash con el que se obtiene la firma digital (PKCS #1) del documento. Su valor puede ser "sha1", "sha256", "sha384" o "sha512 "
estimated_signature_size	Opcional	Opcional	Opcional	Tamaño máximo de la firma expresado en bytes
certification_level	Opcional	Ausente	Opcional	<p>Indicación de si la firma es de certificación:</p> <ul style="list-style-type: none"> • 0: La firma no es de certificación. • 1: La firma es de certificación. Una vez realizada, no se permitirán cambios posteriores en el documento. • 2: La firma es de certificación. Una vez realizada, los únicos cambios que se permitirán en el documento son el rellenado de formularios y la generación de firmas (en campos de firma previamente creados). • 3: La firma es de certificación. Una vez realizada, los únicos cambios que se permitirán en el documento son el rellenado de formularios, la generación de firmas en campos de firma previamente creados y la edición de anotaciones.

contact_info	Opcional	Ausente	Opcional	Información para contactar con el firmante
location	Opcional	Opcional	Opcional	Lugar en el que se firma el documento
reason	Opcional	Opcional	Opcional	Motivo por el que se firma el documento
signature_field	Opcional	Opcional	Opcional	Información del campo de firma
signature_field.name	Opcional	Ausente	Opcional	Nombre del campo de firma. Su valor por defecto es <i>SFLY Signature <seq></i> siendo <i>seq</i> el siguiente número dentro de la secuencia de todos los campos cuyo nombre tiene este formato (al primer campo de esta secuencia le corresponde el número 0)
signature_field.location	Opcional	Opcional	Opcional	Situación del campo de firma dentro del documento
signature_field.location.page	Obligatorio	Obligatorio	Obligatorio	Información de la página en la que se encuentra el campo de firma
signature_field.location.page.number	Obligatorio	Obligatorio	Obligatorio	Número de página en la que se encuentra el campo de firma. El valor "last" denota la última página del documento
signature_field.location.rectangle	Obligatorio	Obligatorio	Obligatorio	Posición y dimensiones del campo de firma
signature_field.location.rectangle.x	Obligatorio	Obligatorio	Obligatorio	Coordenada horizontal de la esquina inferior izquierda del campo de firma
signature_field.location.rectangle.y	Obligatorio	Obligatorio	Obligatorio	Coordenada vertical de la esquina inferior izquierda del campo de firma
signature_field.location.rectangle.height	Obligatorio	Obligatorio	Obligatorio	Altura del campo de firma
signature_field.location.rectangle.width	Obligatorio	Obligatorio	Obligatorio	Anchura del campo de firma
signature_field.appearance	Opcional	Opcional	Opcional	Información sobre la apariencia de la firma
signature_field.appearance.background_image	Obligatorio	Obligatorio	Obligatorio	Imagen de fondo de la apariencia de la firma

signature_field.appearance.background_image.binary	Obligatorio	Obligatorio	Obligatorio	Codificación en Base64 de la imagen de fondo de la apariencia de la firma. Se soportan imágenes JPEG y PNG.
signature_field.appearance.foreground_image	Opcional	Ausente	Opcional	Imagen de primer plano de la apariencia de firma.
signature_field.appearance.foreground_image.binary	Opcional	Ausente	Opcional	Codificación en base64 de la imagen de primer plano de la apariencia de la firma. Se soportan imágenes JPEG y PNG.
signature_field.appearance.signature_details	Opcional	Ausente	Opcional	Información sobre los detalles de la firma que se muestran como texto en su apariencia.
policy_identifier	Variable	Ausente	Variable	Información sobre la política de firma. Esta propiedad es obligatoria si el tipo de firma indicado (propiedad <i>type</i>) es "pades-epes" y debe estar ausente en caso contrario.
policy_identifier.policy_id.oid	Obligatorio	Ausente	Obligatorio	OID que identifica la política de firma
policy_identifier.policy_hash	Obligatorio	Ausente	Obligatorio	Información sobre el hash de la política de firma.
policy_identifier.policy_hash.digest_algorithm_identifier.id	Obligatorio	Ausente	Obligatorio	Algoritmo para calcular el hash de la política de firma. Esta propiedad puede tomar los valores "sha1", "sha256", "sha384" o "sha512"
policy_identifier.policy_hash.digest_value	Obligatorio	Ausente	Obligatorio	Hash de la política de firma.
policy_identifier.policy_qualifiers[]	Obligatorio	Ausente	Obligatorio	Calificadores de la política de firma.
policy_identifier.policy_qualifiers[].type	Obligatorio	Ausente	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor "spuri"
policy_identifier.policy_qualifiers[].uri	Obligatorio	Ausente	Obligatorio	URI de la política de firma.
commitments[]	Opcional	Ausente	Opcional	Compromisos del firmante. Esta propiedad es opcional si el tipo de firma indicado (propiedad <i>type</i>) es "pades-epes" y debe estar ausente en caso contrario.

commitments[].type	Obligatorio	Ausente	Obligatorio	Tipo de uno de los compromisos del firmante: <ul style="list-style-type: none"> • <i>proof_of_origin</i>: el firmante reconoce haber creado y enviado el documento. • <i>proof_of_receipt</i>: el firmante reconoce haber recibido el documento. • <i>proof_of_delivery</i>: el firmante reconoce haber entregado el documento a su destinatario. • <i>proof_of_sender</i>: el firmante reconoce haber enviado el documento. • <i>proof_of_approval</i>: el firmante reconoce que aprueba el documento. • <i>proof_of_creation</i>: el firmante reconoce haber creado.
commitments[].oid	Opcional	Ausente	Opcional	OID que identifica un compromiso del firmante. Solo la tienen los compromisos de tipo <i>custom</i> .

<signature_details>

La propiedad *parameters.signature_field.appearance.signature_details* se amplía en el JSON con la siguiente estructura:

```

{
  "font" : {
    "name" : {string},
    "size" : {number},
    "style" : {string},
    "color" : {
      "r" : {number},
      "g" : {number},
      "b" : {number}
    },
    "encoding" : {string},
    "embed" : boolean
  },
  "details" : [
    {
      "type" : {string},
      "title" : {string},
      "timezone" : {string},
      "format" : {string},
      "value" : {string}
    }
  ]
}

```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
font	Opcional	Ausente	Opcional	Información de la fuente con la que se escriben los detalles de la firma que se muestran como texto en su apariencia.
font.name	Opcional	Ausente	Opcional	Nombre de la fuente con la que se escriben los detalles de la firma que se muestran como texto en su apariencia.
font.size	Opcional	Ausente	Opcional	Tamaño de letra con la que se escriben los detalles de la firma que se muestran como texto en su apariencia.
font.style	Opcional	Ausente	Opcional	Estilo de letra ("normal", "bold", "italic") con el que se escriben los detalles de la firma que se muestran como texto en su apariencia.
font.color	Opcional	Ausente	Opcional	Color con el que se escriben los detalles de la firma que se muestran como texto en su apariencia.
font.color.r	Obligatorio	Ausente	Obligatorio	Nivel de rojo con el que se escriben los detalles de la firma que se muestran como texto en su apariencia (0-255).
font.color.g	Obligatorio	Ausente	Obligatorio	Nivel de verde con el que se escriben los detalles de la firma que se muestran como texto en su apariencia (0- 255).
font.color.b	Obligatorio	Ausente	Obligatorio	Nivel de azul con el que se escriben los detalles de la firma que se muestran como texto en su apariencia (0- 255).

font. embed	Opcional	Ausente	Opcional	Valor booleano (true , false) que indica si la fuente con la que se escriben los detalles de la firma se incluye en el documento firmado.
font. encoding	Opcional	Ausente	Opcional	Codificación ("latin1" , "unicode") de los detalles de la firma que se muestran como texto en su apariencia.
details []	Opcional	Ausente	Opcional	Detalles de la firma que se muestran como texto en su apariencia.
details [].type	Obligatorio	Ausente	Obligatorio	<p>Tipo de un detalle de la firma que se muestra como texto en su apariencia:</p> <ul style="list-style-type: none"> • <i>subject</i>: Titular del certificado delfirmante • <i>name</i>: CN del titular del certificado delfirmante • <i>issuer</i>: emisor del certificado delfirmante • <i>serial</i>: número de serie del certificado delfirmante • <i>contact_info</i>: información para contactar con el firmante (véase la propiedad<i>contact_info</i>) • <i>location</i>: lugar en el que se firma el documento (véase la propiedad<i>location</i>) • <i>reason</i>: motivo por el que se firma el documento (véase la propiedad<i>reason</i>) • <i>date</i>: fecha en la que se firma el documento • <i>text</i>: texto arbitrario
details [].title	Opcional	Ausente	Opcional	Título que precede a un detalle de la firma que se muestra como texto en su apariencia.
details [].timezone	Opcional	Ausente	Opcional	Zona horaria ("GMT" , "Europe/Madrid" , "America/Caracas" , etc.) que se muestra como detalle de la firma en su apariencia. Sólo puede tenerla un detalle de tipo "date".
details [].format	Opcional	Ausente	Opcional	Formato de la fecha que se muestra como detalle de la firma en su apariencia. Sólo puede tenerla un detalle de tipo "date".
details [].value	Opcional	Ausente	Opcional	Texto arbitrario que se muestra como detalle de la firma en su apariencia. Sólo puede tenerla un detalle de tipo "text".

7.3.3.1.2. Firmas XMLDSig/XAdES Enveloped

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type": "document",
    "signature_packaging": "enveloped",
    "nodes_to_sign": [{
      "type": "document_reference",
      "xpath" : {string},
      "uri_type" : {string}
    }],
    "signature_placement": {
      "type" : {string},
      "xpath" : {string}
    }
  },
  "include_data_object_format": {boolean},
  "default_c14n_method": {string},
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	Obligatorio	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> <i>xmlsig</i>: firma XMLDSig <i>xades-bes</i>: firma XadES-BES (Basic Electronic Signature) según ETSI TS 101903 <i>xades-epes</i>: firma XadES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	Ausente	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	Opcional	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	Obligatorio	Obligatorio	Información sobre los datos que se firman, la situación relativa de la firma con respecto a estos datos y la posición de la firma dentro el documento al que se aplica la definición de firma
signature_target.type	Obligatorio	Obligatorio	Obligatorio	Indica si la firma es una contrafirma (countersignature) o no. Actualmente esta propiedad siempre tiene el valor "document" (la firma no es una contrafirma)
signature_target.signature_packaging	Obligatorio	Obligatorio	Obligatorio	Relación de inclusión que tiene la firma respecto a los datos que se firman. Actualmente esta propiedad siempre tiene el valor "enveloped" (la firma está contenida dentro del documento XML firmado)
signature_target	Opcional	Opcional	Opcional	Nodos que se firman
signature_target.nodes_to_sign[].type	Opcional	Opcional	Opcional	Tipo de uno de los nodos que se firman. Actualmente esta propiedad tiene siempre el valor "document_reference" (el nodo pertenece al documento al que se aplica la firma)
signature_target.nodes_to_sign[]	Obligatorio	Obligatorio	Obligatorio	Expresión XPath 2.0 que describe la ruta de uno de los nodos que se firman. Actualmente sólo se soporta la firma del nodo raíz, es decir, siempre tiene el valor "/".

signature_target .nodes_to_sign[] .uri_type	Opcional	Opcional	Opcional	Tipo de URI que se usa para referenciar el nodo que se firma. Su valor puede ser: <ul style="list-style-type: none"> <i>id</i>: Valor por defecto. Identifica el nodofirmado como "" <i>xpointer_xpath</i>: Hace referencia a los datos firmados con "#xpointer(/)"
signature_target .signature_placement	Opcional	Opcional	Opcional	Posición del elemento <i><ds:Signature></i> dentro del documento al que se aplica la firma. El valor por defecto es el " <i>last_child_of</i> " del nodo raíz
signature_target .signature_placement.xpath	Obligatorio	Obligatorio	Obligatorio	Expresión XPath 2.0 que describe (conjuntamente el parámetro <i>signature_target</i> , <i>signature_placement.type</i>) la ruta de la firma dentro del documento al que se aplica la firma
signature_target .signature_placement.type	Obligatorio	Obligatorio	Obligatorio	Relación del nodo que contiene el elemento <i><ds:Signature></i> respecto a la ruta que indica <i>signature_target</i> , <i>signature_placement.xpath</i> . Los valores pueden ser: <ul style="list-style-type: none"> <i>last_child_of</i>: la firma es el último nodo hijo del nodo cuya ruta se indica en la propiedad anterior <i>first_child_of</i>: la firma es el primer nodo hijo del nodo cuya ruta se indica en la propiedad anterior <i>after</i>: la firma es el nodo hermano anterior al nodo cuya ruta se indica en la propiedad anterior <i>before</i>: la firma es el nodo hermano posterior al nodo cuya ruta se indica en la propiedad anterior
include_data_object_format	Opcional	Opcional	Opcional	Esta propiedad es opcional e indica (true, false) si en la firma debe indicarse el tipo de los datos firmados (elemento <i><xades:DataObjectFormat></i>). El valor por defecto es true y el formato que se indica es <i>text/xml</i>

default_c14n_method	Opcional	Opcional	Opcional	<p>Algoritmo de canonicalización que se utiliza:</p> <ul style="list-style-type: none"> • <i>c14n</i>: algoritmo inclusivo • <i>c14nWithComments</i>: algoritmo inclusivo con comentarios) • <i>excC14n</i>: algoritmo exclusivo • <i>excC14nWithComments</i>: algoritmo exclusivo con comentarios
policy_identifier	Opcional	Opcional	Opcional	Información sobre la política de firma. Si la firma es <i>xades-epes</i> la propiedad es obligatoria.
policy_identifier.policy_identifier.uri	Obligatorio	Obligatorio	Obligatorio	URI de la política de firma.
policy_identifier.policy_identifier.qualifier	Opcional	Opcional	Opcional	Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son: <ul style="list-style-type: none"> • OIDsAsURI • OIDsAsURN
policy_identifier.policy_id.description	Opcional	Opcional	Opcional	Descripción de la política.
policy_identifier.policy_id.documentation	Opcional	Opcional	Opcional	Documentos con información adicional sobre la política.
policy_identifier.policy_id.documentation_uri	Obligatorio	Obligatorio	Obligatorio	URI de un documento con información adicional sobre la política.
policy_identifier.policy_hash	Obligatorio	Obligatorio	Obligatorio	Información sobre el hash de la política de firma.
policy_identifier.policy_hash.digest_algorithm	Obligatorio	Obligatorio	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"

policy_identifier .policy_hash.digest_value	Obligatorio	Obligatorio	Obligatorio	Hash de la política de firma.
policy_identifier .policy_qualifiers[]	Opcional	Opcional	Opcional	Calificadores de la política de firma.
policy_identifier .policy_qualifiers[].type	Obligatorio	Obligatorio	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor " <i>spuri</i> ".
policy_identifier .policy_qualifiers[].uri	Obligatorio	Obligatorio	Obligatorio	URI de uno de los calificadores de la política de firma.

7.3.3.1.3. Firmas XMLDSig/XAdES Enveloping

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type": "document",
    "signature_packaging": "enveloping"
  },
  "include_data_object_format" : {boolean},
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	No disponible	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> <i>xmlsig</i>: firma XMLDSig <i>xades-bes</i>: firma XadES-BES (Basic Electronic Signature) según ETSI TS 101903 <i>xades-epes</i>: firma XadES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	No disponible	Opcional	Certificado del firmante codificado en Base64.
default_digest_algorithm	Opcional	No disponible	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto.
signature_target	Obligatorio	No disponible	Obligatorio	Información sobre los datos que se firman y la posición relativa de la firma con respecto a estos datos.
signature_target.type	Obligatorio	No disponible	Obligatorio	Indica si la firma es una contrafirma (countersignature) o no. Esta propiedad es opcional y solo puede tener el valor "document" (que es el valor que toma por defecto).
signature_target.signature_packaging	Obligatorio	No disponible	Obligatorio	Posición relativa de la firma respecto a los datos que se firman. Siempre tiene el valor "enveloping" (la firma XML contiene los datos que se firman).
include_data_object_format	Opcional	No disponible	Opcional	Esta propiedad es opcional e indica (true, false) si en la firma debe indicarse el tipo de los datos firmados (elemento <i><xades:DataObjectFormat></i>). El valor por defecto es true y el formato que se indica es <i>text/xml</i>
policy_identifier	Opcional	No disponible	Opcional	Información sobre la política de firma. Si la firma es <i>xades-epes</i> la propiedad es obligatoria

policy_identifier .policy_id.identifier.uri	Obligatorio	No disponible	Obligatorio	URI de la política de firma
policy_identifier .policy_id.identifier.qualifier	Opcional	No disponible	Opcional	Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son: <ul style="list-style-type: none"> • OIDAURI • OIDAURN
policy_identifier .policy_id.description	Opcional	No disponible	Opcional	Descripción de la política
policy_identifier .policy_id.documentation_references[]	Opcional	No disponible	Opcional	Documentos con información adicional sobre la política
policy_identifier .policy_id.documentation_	Obligatorio	No disponible	Obligatorio	URI de un documento con información adicional sobre la política
policy_identifier .policy_hash	Obligatorio	No disponible	Obligatorio	Información sobre el hash de la política de firma
policy_identifier .policy_hash.digest_algorithm	Obligatorio	No disponible	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"
policy_identifier .policy_hash.	Obligatorio	No disponible	Obligatorio	Hash de la política de firma
policy_identifier .policy_qualifiers[]	Opcional	No disponible	Opcional	Calificadores de la política de firma

policy_identifier.policy_qualifiers[].type	Obligatorio	No disponible	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor " <i>spuri</i> "
policy_identifier.policy_qualifiers[].uri	Obligatorio	No disponible	Obligatorio	URI de uno de los calificadores de la política de firma

7.3.3.1.4. Firmas XMLDSig/XAdES

Enveloping con elemento Manifest

La propiedad *parameters* se representa en el JSON del siguiente modo

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string},
  "signature_target" : {
    "type": "document",
    "signature_packaging": "enveloping",
    "nodes_to_sign": [
      {
        "type": "manifest",
        "references": [
          {
            "uri": {string},
            "digest_algorithm": {string},
            "digest_value": {string},
            "transforms": [
              {
                "type": "c14n",
                "method": {string}
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	No disponible	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> • <i>xmldsig</i>: firma XMLDSig • <i>xades-bes</i>: firma XadES-BES (Basic Electronic Signature) según ETSI TS 101903 • <i>xades-epes</i>: firma XadES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	No disponible	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	No disponible	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	No disponible	Obligatorio	Información sobre los datos que se firman y la posición relativa de la firma con respecto a estos datos

<code>signature_target.type</code>	Obligatorio	No disponible	Obligatorio	Esta propiedad es opcional y solo puede tener el valor "document" (que es el valor que toma por defecto)
<code>signature_target.signature_packaging</code>	Obligatorio	No disponible	Obligatorio	Posición relativa de la firma respecto a los datos que se firman. Siempre tiene el valor "enveloping" (la firma XML contiene el elemento <i>Manifest</i> que se firma)
<code>signature_target.nodes_to_sign[]</code>	Obligatorio	No disponible	Obligatorio	Esta propiedad contiene información sobre los datos que se firman. El array solo puede tener información de un elemento.
<code>signature_target.nodes_to_sign[0].type</code>	Obligatorio	No disponible	Obligatorio	Esta propiedad solo puede tener el valor "manifest"
<code>signature_target.nodes_to_sign[0].references[]</code>	Obligatorio	No disponible	Obligatorio	Información sobre los datos que se firman mediante su inclusión en el Manifest. Puede haber de 1 a N referencias.
<code>signature_target.nodes_to_sign[0].references[].uri</code>	Obligatorio	No disponible	Obligatorio	URI de uno de los datos que se firman mediante su inclusión en el Manifest.
<code>signature_target.nodes_to_sign[0].references[].digest_algorithm</code>	Obligatorio	No disponible	Obligatorio	Algoritmo con el que se ha obtenido el hash de uno de los datos que se firman mediante su inclusión en el Manifest. Su valor puede ser "sha1", "sha256", "sha384" o "sha512".
<code>signature_target.nodes_to_sign[0].references[].digest_value</code>	Obligatorio	No disponible	Obligatorio	Hash de uno de los datos que se firman mediante su inclusión en el Manifest.
<code>signature_target.nodes_to_sign[0].references[].transforms[]</code>	Opcional	No disponible	Opcional	Esta propiedad es opcional y contiene información sobre las transformaciones realizadas sobre uno de los datos que se firman mediante su inclusión en el Manifest, antes de calcular su hash (el array sólo tiene un elemento).

<code>signature_target.nodes_to_sign[0].references[].transforms[0].type</code>	Obligatorio	No disponible	Obligatorio	Esta propiedad es obligatoria e indica el tipo de una de las transformaciones realizadas sobre uno de los datos que se firman mediante su inclusión en el <i>Manifest</i> , antes de calcular su hash. Actualmente solo se permite el valor <i>c14n</i> (las únicas transformaciones soportadas son las de <i>canonicalización</i>).
<code>signature_target.nodes_to_sign[0].references[].transforms[0].method</code>	Obligatorio	No disponible	Obligatorio	Esta propiedad es obligatoria e indica una de las transformaciones realizadas sobre uno de los datos que se firman mediante su inclusión en el <i>Manifest</i> , antes de calcular su hash. Los valores que puede tener esta propiedad corresponden todos a algoritmos de <i>canonicalización</i> : <ul style="list-style-type: none"> • <i>c14n</i>: algoritmo inclusivo (http://www.w3.org/TR/2001/REC-xml-c14n-20010315) • <i>c14nWithComments</i>: algoritmo inclusivo con comentarios (http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments) • <i>excC14n</i>: algoritmo exclusivo (http://www.w3.org/2001/10/xml-exc-c14n#) • <i>excC14nWithComments</i>: algoritmo exclusivo con comentarios (http://www.w3.org/2001/10/xml-exc-c14n#WithComments)

Consideraciones a tener en cuenta sobre el elemento Manifest:

1. Lo único que diferencia una firma XAdES Enveloping regular/ tradicional de una con elemento Manifest, a nivel de sintaxis, es el nodo `<Reference>` incluido en el *SignedInfo*. Mientras que en la primera el atributo URI apunta directamente al nodo `<Object>`, en la segunda apunta al propio nodo `<Manifest>` incluido dentro del nodo `<Object>`, además de que debe incluir un atributo *Type* específico que indique que la referencia firmada es un Manifest.

XAdES Enveloping

```
<dsig:Signature>
  <dsig:SignedInfo>
    [...]
    dsig:Reference Id="r-id-1" Type="http://www.w3.org/2000/09/xmldsig#Object"
    URI="#o-id-1">
      <dsig:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
      <dsig:DigestValue>XGt...UqUWA=</dsig:DigestValue>
    </dsig:Reference>
  </dsig:SignedInfo>
  [...]
  <dsig:Object Encoding=http://www.w3.org/2000/09/xmldsig#base64 Id="o-id-1">
    IDxpb...1lbnRvPg==</dsig:Object>
  [...]
</dsig:Signature>
```

XAdES Enveloping con elemento Manifest

```
<dsig:Signature>
  <dsig:SignedInfo>
    [...]
    <dsig:Reference Id="Id149746743018292238141588140869"
      Type="http://www.w3.org/2000/09/xmldsig#Manifest" URI="#7408bc2a-8928-40e5-a818-
      db070fdd58f0">
      <dsig:Transforms>
        <dsig:Transform Algorithm="http://www.w3.org/TR/2001/REC-
        xml-c14n-20010315"/>
      </dsig:Transforms>
      <dsig:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
      <dsig:DigestValue>IfE...OS4bltw==</dsig:DigestValue>
    </dsig:SignedInfo>
    [...]
    <dsig:Object>
      <ds:Manifest xmlns:ds=http://www.w3.org/2000/09/xmldsig# Id="7408bc2a-
      8928-40e5-a818-db070fdd58f0">
        <ds:Reference URI="id-manifest:1" Id="f97861f0-fa0f-47df-87cb-
        e5244d7c58fa" Type="http://www.w3.org/2000/09/xmldsig#Object">
          <ds:DigestMethod Algorithm="
          http://www.w3.org/2001/04/xmlenc#sha512"/>
          <ds:DigestValue> Uef...PCJFQdQRQ==</ds:DigestValue>
        </ds:Reference>
      </ds:Manifest>
    </dsig:Object>
    [...]
</dsig:Signature>
```

2. Un elemento Manifest debe calcularse utilizando el digest más seguro.
3. Un Manifest debe considerarse un mecanismo donde la responsabilidad y la validez del *digest* están fuera del ámbito de la firma. El propietario es quien proporciona el elemento Manifest, y debe decidir si tiene sentido o no según los requisitos de su negocio.
4. A la hora de generar firmas XAdES Enveloping con varios elementos Manifest, hay que tener en cuenta que el servicio de firma de Zain no utiliza la misma implementación subyacente que la API de firma de documentos de Giltza, utiliza una más antigua. Por lo tanto, a la hora de realizar firmas con varios elementos Manifest, Zain generará un nodo Manifest por cada uno de ellos y Giltza generará un único nodo Manifest con una referencia por cada uno de ellos.

Giltza – Firma XAdES Enveloping con varios elementos Manifest

```
<dsig:Signature>
  <dsig:SignedInfo>
    [...]
    <dsig:Reference Id="f2619d4a-425d-4c52-9658-d0fc09281967"
Type="http://www.w3.org/2000/09/xmldsig#Manifest" URI="#e1829b27-f525-49cd-b955-4e3f5718c91f">
      <dsig:Transforms>
        <dsig:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      </dsig:Transforms>
      <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmenc#sha512"/>
      <dsig:DigestValue>uKa...HChJQ==</dsig:DigestValue>
    </dsig:Reference>
  </dsig:SignedInfo>
  [...]
  <dsig:Object>
    <dsig:Manifest Id="e1829b27-f525-49cd-b955-4e3f5718c91f">
      <dsig:Reference Id="6f26550e-1d28-43aa-9e78-de21dc5d100b"
Type="http://www.w3.org/2000/09/xmldsig#Object" URI="id-manifest:1">
        <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmenc#sha512"/>
        <dsig:DigestValue>UeO...dQRQ==</dsig:DigestValue>
        <dsig:Transforms>
          <dsig:Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </dsig:Transforms>
      </dsig:Reference>
      <dsig:Reference Id="b5b5ab55-9580-4fb7-9f9d-6d11d38c1d91"
Type="http://www.w3.org/2000/09/xmldsig#Object" URI="id-manifest:2">
        <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmenc#sha512"/>
        <dsig:DigestValue>qZD...t4A7Dw==</dsig:DigestValue>
      </dsig:Reference>
    </dsig:Manifest>
  </dsig:Object>
  [...]
</dsig:Signature>
```

Zain – Firma XAdES Enveloping con varios elementos Manifest

```
<dsig:Signature>
  <dsig:SignedInfo>
    [...]
    <dsig:Reference Id="Id7964458016500874462097861201"
Type="http://www.w3.org/2000/09/xmldsig#Manifest" URI="#Manifest1">
      <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
      <dsig:DigestValue>Gii...DQA=</dsig:DigestValue>
    </dsig:Reference>
    <dsig:Reference Id="Id8395754253796159331661619243"
Type="http://www.w3.org/2000/09/xmldsig#Manifest" URI="#Manifest2">
      <dsig:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
      <dsig:DigestValue>ba2...hiv2aU=</dsig:DigestValue>
    </dsig:Reference>
  </dsig:SignedInfo>
  [...]
  <dsig:Object Id="id-object-1">
    <ds:Manifest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
Id="Manifest1">
      <ds:Reference URI="id-manifest:1">
        <ds:DigestMethod
```

```

Algorithm="http://www.w3.org/2001/04/xmenc#sha512"
/>
    <ds:DigestValue>uGX...3u6h/c=</ds:DigestValue>
  </ds:Reference>
</ds:Manifest>
</ds:Object>
<dsig:Object Id="id-object-2">
  <ds:Manifest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Id="Manifest2">
    <ds:Reference URI="id-manifest:2">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmenc#sha512"
      />
      <ds:DigestValue>9cE...cJhKlM=</ds:DigestValue>
    </ds:Reference>
  </ds:Manifest>
</ds:Object>
[...]
```

5. A la hora de verificar en Zain firmas XAdES Enveloping con elemento Manifest, la implementación se va a realizar del mismo modo, tanto si la firma se ha generado mediante Giltza o a través de Zain. Consultar el apartado [13. Ejemplos de verificación de firmas mediante Zain](#), para más información.
6. Respecto a las firmas múltiples, es posible añadir firmas nuevas a firmas ya existentes XAdES Enveloping con elemento Manifest:
 - a. Una firma XAdES "paralela" siempre se puede hacer sobre cualquier conjunto de datos, ya que se puede colocar donde se quiera, no hay relación real con ninguna firma existente.
 - b. Una firma "en serie" XAdES es una contrafirma, que cubre la firma existente y puede ser compatible con una firma que contenga elemento Manifest o no.

7.3.3.1.5. Firmas XMLDSig/XAdES Internally Detached

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type": "document", "
    signature_packaging": "detached",
    "nodes_to_sign": [{
      "type": "document_reference",
      "xpath" : {string}
    }],
    "signature_placement": {
      "type" : {string},
      "xpath" : {string}
    }
  },
  "include_data_object_format": {48boolean},
  "default_c14n_method": {string},
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	Obligatorio	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> <i>xmlsig</i>: firma XMLDSig <i>xades-bes</i>: firma XadES-BES (Basic Electronic Signature) según ETSI TS 101903 <i>xades-epes</i>: firma XadES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	Ausente		Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	Opcional		Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	Obligatorio	Obligatorio	Información sobre los datos que se firman, la situación relativa de la firma con respecto a estos datos y la posición de la firma dentro del documento al que se aplica la definición de firma
signature_target.type	Obligatorio	Obligatorio	Obligatorio	Indica si la firma es una contrafirma (countersignature) o no. Actualmente esta propiedad siempre tiene el valor "document" (la firma no es una contrafirma)
signature_target.signature_packaging	Obligatorio	Obligatorio	Obligatorio	Relación de inclusión que tiene la firma respecto a los datos que se firman. Actualmente esta propiedad siempre tiene el valor "detached" (la firma está separada de los datos que se firman)
signature_target	Obligatorio	Obligatorio	Obligatorio	Nodos que se firman
signature_target.nodes_to_sign[]	Obligatorio	Obligatorio	Obligatorio	Tipo de uno de los nodos que se firman. Actualmente esta propiedad siempre tiene el valor "document_reference" (el nodo pertenece al documento al que se aplica la firma)
signature_target.nodes_to_sign[]	Obligatorio	Obligatorio	Obligatorio	Expresión XPath 2.0 que describe la ruta de uno de los nodos que se firman. Actualmente sólo se soporta la firma del nodo raíz, es decir, siempre tiene el valor "/".

signature_target.signature_placement	Opcional	Opcional	Opcional	Posición del elemento <i><ds:Signature></i> dentro del documento al que se aplica la firma. El valor por defecto es el " <i>last_child_of</i> " del nodo raíz
signature_target.signature_placement.xpath	Obligatorio	Obligatorio	Obligatorio	Expresión XPath 2.0 que describe (conjuntamente con el parámetro <i>signature_target.signature_placement.type</i>) la ruta de la firma dentro del documento al que se aplica la firma
signature_target.signature_placement.type	Obligatorio	Obligatorio	Obligatorio	Relación del nodo que contiene el elemento <i><ds:Signature></i> respecto a la ruta que indica <i>signature_target.signature_placement.xpath</i> . Los valores pueden ser: <ul style="list-style-type: none"> • <i>last_child_of</i>: la firma es el último nodo hijo del nodo cuya ruta se indica en la propiedad anterior • <i>first_child_of</i>: la firma es el primer nodo hijo del nodo cuya ruta se indica en la propiedad anterior • <i>after</i>: la firma es el nodo hermano anterior al nodo cuya ruta se indica en la propiedad anterior • <i>before</i>: la firma es el nodo hermano posterior al nodo cuya ruta se indica en la propiedad anterior
include_data_object_format	Opcional	Opcional	Opcional	Esta propiedad es opcional e indica (true, false) si en la firma debe indicarse el tipo de los datos firmados (elemento <i><xades:DataObjectFormat></i>). El valor por defecto es true y el formato que se indica es <i>text/xml</i>

default_c14n_method	Opcional	Opcional	Opcional	<p>Algoritmo de <i>canonicalización</i> que se utiliza:</p> <ul style="list-style-type: none"> • <i>c14n</i>: algoritmo inclusivo (http://www.w3.org/TR/2001/REC-xml-c14n-20010315) • <i>c14nWithComments</i>: algoritmo inclusivo con comentarios (http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments) • <i>excC14n</i>: algoritmo exclusivo (http://www.w3.org/2001/10/xml-exc-c14n#) • <i>excC14nWithComments</i>: algoritmo exclusivo con comentarios (http://www.w3.org/2001/10/xml-exc-c14n#WithComments)
policy_identifier	Opcional	Opcional	Opcional	Información sobre la política de firma. Si la firma es <i>xades-epes</i> la propiedad es obligatoria
policy_identifier.policy_identifier.uri	Obligatorio	Obligatorio	Obligatorio	URI de la política de firma
policy_identifier.policy_identifier.qualifier	Opcional	Opcional	Opcional	<p>Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son:</p> <ul style="list-style-type: none"> • OIDAURI • OIDAURN
policy_identifier.policy_id.description	Opcional	Opcional	Opcional	Descripción de la política.
policy_identifier.policy_id.documentation	Opcional	Opcional	Opcional	Documentos con información adicional sobre la política.

policy_identifier .policy_id. documentation_ references[].u ri	Obligatorio	Obligatorio	Obligatorio	URI de un documento con información adicional sobre la política.
policy_identifier	Obligatorio	Obligatorio	Obligatorio	Información sobre el hash de la política de firma.
policy_identifier .policy_hash. digest_algori th m_identifier. id	Obligatorio	Obligatorio	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", " sha384" o "sha512"
policy_identifier .policy_has h.	Obligatorio	Obligatorio	Obligatorio	Hash de la política de firma.
policy_identifier .policy_qualifi er s[]	Opcional	Opcional	Opcional	Calificadores de la política de firma.
policy_identifier .policy_qualifi er s[]. type	Obligatorio	Obligatorio	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor " <i>spuri</i> ".
policy_identifier .policy_qualifi er s[].uri	Obligatorio	Obligatorio	Obligatorio	URI de uno de los calificadores de la política de firma.

7.3.3.1.6. Firmas XMLDSig/XAdES Externally Detached

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type" : "document",
    "signature_packaging" : "detached",
    "nodes_to_sign": [{
      "type" : "raw_reference",
      "uri" : {string}
    }]
  },
  "include_data_object_format" : {boolean},
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki<	Idazki >= 3.0	Descripción
type	Obligatorio	Obligatorio	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> • <i>xmlsig</i>: firma XMLDSig • <i>xades-bes</i>: firma XAdES-BES (Basic Electronic Signature) según ETSI TS 101903 • <i>xades-epes</i>: firma XAdES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	Ausente	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	Opcional	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	Obligatorio	Obligatorio	Información sobre los datos que se firman, la situación relativa de la firma con respecto a estos datos y la posición de la firma dentro del documento al que se aplica la definición de firma
signature_target.type	Obligatorio	Obligatorio	Obligatorio	Indica si la firma es una contrafirma (countersignature) o no. Actualmente esta propiedad siempre tiene el valor "document" (la firma no es una contrafirma)
signature_target.signature_packaging	Obligatorio	Obligatorio	Obligatorio	Relación de inclusión que tiene la firma respecto a los datos que se firman. Actualmente esta propiedad siempre tiene el valor "detached" (la firma está separada de los datos que se firman)
signature_target	Opcional	Opcional	Opcional	Nodos que se firman
signature_target.nodes_to_sign[]	Opcional	Opcional	Opcional	Tipo de datos que se firman. Actualmente esta propiedad tiene siempre el valor "raw_reference" (los datos están dentro del documento, de cualquier tipo, al que se aplica la firma)

signature_target .nodes_to_sign[] .uri	Obligatorio	Obligatorio	Obligatorio	URI de los datos que se firman. Este parámetro es opcional y su valor por defecto es "file:///<docname>", siendo <docname>el nombre del documento al que se aplica la definición de firma. Este nombre es el que tiene la parte document del mensaje multipart mediante el que se crea el proceso de firma
include_data_object_format	Opcional	Opcional	Opcional	Indica (true, false) si en la firma debe indicarse el tipo de los datos firmados (<i>DataObjectFormat</i>). El valor por defecto es true y el formato que se indica es el tipo MIME proporcionado al crear el documento al que se aplica la definición de firma
policy_identifier	Opcional	Opcional	Opcional	Información sobre la política de firma. Si la firma es <i>xades-epes</i> la propiedad es obligatoria
policy_identifier .policy_id.identifier.uri	Obligatorio	Obligatorio	Obligatorio	URI de la política de firma
policy_identifier .policy_id.identifier.qualifier	Opcional	Opcional	Opcional	Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son: <ul style="list-style-type: none"> • OIDsAsURI • OIDsAsURN
policy_identifier .policy_id.description	Opcional	Opcional	Opcional	Descripción de la política
policy_identifier .policy_id.documentation_references[]	Opcional	Opcional	Opcional	Documentos con información adicional sobre la política
policy_identifier .policy_id.documentation_references[].uri	Obligatorio	Obligatorio	Obligatorio	URI de un documento con información adicional sobre la política

policy_identifier.policy_hash	Obligatorio	Obligatorio	Obligatorio	Información sobre el hash de la política de firma
policy_identifier.policy_hash.digest_algorithm_identifier.id	Obligatorio	Obligatorio	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"
policy_identifier.policy_hash.digest_value	Obligatorio	Obligatorio	Obligatorio	Hash de la política de firma
policy_identifier.policy_qualifiers[]	Opcional	Opcional	Opcional	Calificadores de la política de firma
policy_identifier.policy_qualifiers[].type	Obligatorio	Obligatorio	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor " <i>spuri</i> "
policy_identifier.policy_qualifiers[].uri	Obligatorio	Obligatorio	Obligatorio	URI de uno de los calificadores de la política de firma

7.3.3.1.7. Firmas XMLDSig/XAdES Externally Detached a partir de un hash

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type" : "document",
    "signature_packaging" : "detached"
    "nodes_to_sign": [{
      "type" : "external_reference",
      "uri" : {string},
      "reference_type" : {string}
      "digest_algorithm" : {string},
      "digest_value" : {string},
      "transforms" : [{
        "type" : "c14n",
        "method" : {string}
      }]
    }]
  },
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	Obligatorio	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> <i>xmldsig</i>: firmaXMLDSig <i>xades-bes</i>: firma XAdES-BES (Basic Electronic Signature) según ETSI TS 101903 <i>xades-epes</i>: firma XAdES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	Ausente	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	Opcional	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	Obligatorio	Obligatorio	Información sobre los datos que se firman, la situación relativa de la firma con respecto a estos datos y la posición de la firma dentro el documento al que se aplica la definición de firma
signature_target.type	Obligatorio	Obligatorio	Obligatorio	Indica si la firma es una contrafirma (countersignature) o no. Actualmente esta propiedad siempre tiene el valor " <i>document</i> " (la firma no es una contrafirma).
signature_target.signature_packaging	Obligatorio	Obligatorio	Obligatorio	Relación de inclusión que tiene la firma respecto a los datos que se firman. Actualmente esta propiedad siempre tiene el valor " <i>detached</i> " (la firma está separada de los datos que se firman).
signature_target	Opcional	Opcional	Opcional	Nodos que se firman.
signature_target.nodes_to_sign[].type	Opcional	Opcional	Opcional	Tipo de datos que se firman. Actualmente esta propiedad tiene siempre el valor " <i>external_reference</i> " (los datos son externos, no se encuentran en la plataforma)

signature_target .nodes_to_sign[] .uri	Obligatorio	Obligatorio	Obligatorio	URI de uno de los datos que se firman que deberá constar en la firma.
signature_target .nodes_to_sign[] .digest_algorithm	Obligatorio	Obligatorio	Obligatorio	Algoritmo con el que se ha obtenido el hash de uno de los datos que se firman. Su valor puede ser “sha1”, “sha256”, “sha384” o “sha512”.
signature_target .nodes_to_sign[] .digest_value	Obligatorio	Obligatorio	Obligatorio	<i>Hash</i> de uno de los datos que se firman.
signature_target .nodes_to_sign[] .reference_type	Opcional	Opcional	Opcional	Contiene información del tipo de los datos firmados (corresponde al atributo <i>Type</i> del elemento <i><Reference></i>)
signature_target .nodes_to_sign[] .transforms[]	Opcional	Opcional	Opcional	Contiene información sobre las transformaciones realizadas sobre los datos que se firman antes de calcular su hash.
signature_target .nodes_to_sign[] .transforms[].type	Obligatorio	Obligatorio	Obligatorio	Indica el tipo de una de las transformaciones realizadas sobre los datos que se firman antes de calcular su hash. Actualmente, sólo se permite el valor <i>c14n</i>
signature_target .nodes_to_sign[] .transforms[].method	Obligatorio	Obligatorio	Obligatorio	Indica una de las transformaciones realizadas sobre los datos que se firman antes de calcular su <i>hash</i> :

policy_identifier	Opcional	Opcional	Opcional	Información sobre la política de firma. Si la firma es <i>xades-epes</i> la propiedad es obligatoria.
policy_identifier.policy_identifier.uri	Obligatorio	Obligatorio	Obligatorio	URI de la política de firma.
policy_identifier.policy_identifier.qualifier	Opcional	Opcional	Opcional	Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son: <ul style="list-style-type: none"> • OIDsURI • OIDsURN
policy_identifier.policy_id.description	Opcional	Opcional	Opcional	Descripción de la política.
policy_identifier.policy_id.documentation_references[]	Opcional	Opcional	Opcional	Documentos con información adicional sobre la política.
policy_identifier.policy_id.documentation_references[].uri	Obligatorio	Obligatorio	Obligatorio	URI de un documento con información adicional sobre la política.
policy_identifier.policy_hash	Obligatorio	Obligatorio	Obligatorio	Información sobre el hash de la política de firma.
policy_identifier.policy_hash.digest_algorithm_identifier.id	Obligatorio	Obligatorio	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"

policy_identifier .policy_hash.digest_value	Obligatorio	Obligatorio	Obligatorio	Hash de la política de firma.
policy_identifier .policy_qualifiers[]	Opcional	Opcional	Opcional	Calificadores de la política de firma.
policy_identifier .policy_qualifiers[].type	Obligatorio	Obligatorio	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor " <i>spuri</i> ".
policy_identifier .policy_qualifiers[].uri	Obligatorio	Obligatorio	Obligatorio	URI de uno de los calificadores de la política de firma.

7.3.3.1.8. Contrafirmas XAdES

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate": {string},
  "default_digest_algorithm": {string}
  "signature_target":
  {
    "type" : "signature",
    "signature_packaging" : "string"
    "nodes_to_sign": [{
      "type" : "signature_reference",
      "xpath" : {string}
    }],
    "signature_placement": {
      "type" : "string",
      "xpath" : {string}
    }
  },
  "default_c14n_method" : {string},
  "policy_identifier" : {
    "policy_id" : {
      "identifier" : {
        "uri" : {string},
        "qualifier" : {string}
      },
      "description" : {string},
      "documentation_references" : [{
        "uri" : {string}
      }]
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    },
    "policy_qualifiers" : [{
      "type" : "spuri",
      "uri" : {string}
    }]
  }
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	No disponible	Obligatorio	Tipo de firma XML. Los valores soportados son: <ul style="list-style-type: none"> • <i>xmlsig</i>: firma XMLDSig • <i>xades-bes</i>: firma XAdES-BES (Basic Electronic Signature) según ETSI TS 101903 • <i>xades-epes</i>: firma XAdES-EPES (Explicit Policy Electronic Signature) según ETSI TS 101903
certificate	Opcional	No disponible	Opcional	Certificado del firmante codificado en Base64.
default_digest_algorithm	Opcional	No disponible	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto
signature_target	Obligatorio	No disponible	Obligatorio	Información sobre la firma que se contrafirma y la posición relativa de la firma con respecto a estos datos
signature_target.type	Obligatorio	No disponible	Obligatorio	Esta propiedad solo puede tener el valor "signature" y sirve para indicar que los datos a firmar corresponden a una firma que se desea contrafirmar.
signature_target.signature_packaging	Obligatorio	No disponible	Obligatorio	Posición relativa de la contrafirma respecto a la firma que se contrafirma: <ul style="list-style-type: none"> • <i>enveloped</i>: la contrafirma está contenida dentro de la firma XML que se contrafirma. • <i>detached</i>: la contrafirma está separada de la firma XML que se contrafirma (en consecuencia, se podrá actualizar la contrafirma después de añadir un sello de tiempo de archivo a la firma que se contrafirma).

<code>signature_target.nodes_to_sign[]</code>	Opcional	No disponible	Opcional	<p>Contiene información de la firma o firmas que se quieren contrafirmar:</p> <ul style="list-style-type: none"> • <i>enveloped</i>: el <i>array</i> solo puede tener un elemento. Por defecto, la contrafirma se aplicará a la única firma (o contrafirma) que no haya sido contrafirmada. Si más de una firma (o ninguna) cumple esta condición, se producirá un error cuando se realice la firma. • <i>Detached</i>: el <i>array</i> puede tener más de un elemento, cada uno apuntando a una firma diferente. En la contrafirma resultante se creará un elemento <i>Reference</i> por cada elemento que se haya indicado. Se requiere que cada una de las firmas que se quieran contrafirmar incluya un elemento <i>SignatureValue</i> con el atributo <i>Id</i>.
<code>signature_target.nodes_to_sign[].type</code>	Opcional	No disponible	Opcional	Tipo de referencia con la que se remite a la firma que se contrafirma. Esta propiedad solo puede tener el valor "signature_reference".
<code>signature_target.nodes_to_sign[].xpath</code>	Obligatorio	No disponible	Obligatorio	Expresión XPath 2.0 que describe la ruta de la firmas que se contrafirman (una en el caso "enveloped", cualquier número en el caso "detached"). La expresión tiene que resolverse en uno o varios elementos <i>Signature</i> dichos elementos deben tener el atributo <i>Id</i> .
<code>signature_target.signature_placement</code>	Opcional	No disponible	Opcional	Contiene información sobre la posición de la contrafirma (del elemento <code><ds:Signature></code>) dentro del documento al que se aplica la definición de firma. Si no se incluye, cuando el proveedor de firma realice la contrafirma en base a la definición de firma, devolverá sólo la contrafirma en cuestión.

signature_target.signature_placement.xpath	Obligatorio	No disponible	Obligatorio	Expresión XPath 2.0 que describe (conjuntamente el parámetro <i>signature_target.signature_placement.type</i>) la ruta de la firma dentro del documento al que se aplica la firma.
signature_target.signature_placement.type	Obligatorio	No disponible	Obligatorio	Relación del nodo que contiene el elemento <code><ds:Signature></code> respecto a la ruta que indica <i>signature_target.signature_placement.xpath</i> . Los valores pueden ser: <ul style="list-style-type: none"> • <i>last_child_of</i>: la firma es el último nodo hijo del nodo cuya ruta se indica en la propiedad anterior. • <i>first_child_of</i>: la firma es el primer nodo hijo del nodo cuya ruta se indica en la propiedad anterior. • <i>after</i>: la firma es el nodo hermano anterior al nodo cuya ruta se indica en la propiedad anterior. • <i>before</i>: la firma es el nodo hermano posterior al nodo cuya ruta se indica en la propiedad anterior.
default_c14n_method	Opcional	No disponible	Opcional	Algoritmo de canonicalización que se utiliza: <ul style="list-style-type: none"> • <i>c14n</i>: algoritmo inclusivo • <i>c14nWithComments</i>: algoritmo inclusivo con comentarios • <i>excC14n</i>: algoritmo exclusivo • <i>excC14nWithComments</i>: algoritmo exclusivo con comentarios Su valor por defecto es: <i>excC14n</i>
policy_identifier	Opcional	No disponible	Opcional	Información sobre la política de firma. Si la firma es xades-epes la propiedad es obligatoria.

policy_identifier .policy_id.identifier.uri	Obligatorio	No disponible	Obligatorio	URI de la política de firma.
policy_identifier .policy_id.identifier.qualifier	Opcional	No disponible	Opcional	Tipo de URI de la política de firma cuando dicha URI se construye partir de un OID. Sus valores posibles son: <ul style="list-style-type: none"> • OIDsURI • OIDsURN
policy_identifier .policy_id.description	Opcional	No disponible	Opcional	Descripción de la política.
policy_identifier .policy_id.documentations_references[]	Opcional	No disponible	Opcional	Documentos con información adicional sobre la política.
policy_identifier .policy_id.documentations_references[].uri	Obligatorio	No disponible	Obligatorio	URI de un documento con información adicional sobre la política.
policy_identifier .policy_hash	Obligatorio	No disponible	Obligatorio	Información sobre el hash de la política de firma.
policy_identifier .policy_hash.digest_algorithm_identifier.id	Obligatorio	No disponible	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"
policy_identifier .policy_hash.digest_value	Obligatorio	No disponible	Obligatorio	Hash de la política de firma

policy_identifier .policy_qualifiers[]	Opcional	No disponible	Opcional	Calificadores de la política de firma.
policy_identifier .policy_qualifiers[]. type	Obligatorio	No disponible	Obligatorio	Tipo de uno de los calificadores de la política de firma. Esta propiedad tiene siempre el valor "spuri".
policy_identifier .policy_qualifiers[].uri	Obligatorio	No disponible	Obligatorio	URI de uno de los calificadores de la política de firma.

7.3.3.1.9 Firmas CMS/CADES

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate" : {string},
  "default_digest_algorithm" : {string}
  "signature_target":
  {
    "type" : "document",
    "signature_packaging" : {string},
  },
  "policy_identifier" : {
    "policy_id" : {
      "oid" : {string}
    },
    "policy_hash" : {
      "digest_algorithm_identifier" : {
        "id" : {string}
      },
      "digest_value" : {string}
    }
  },
  "mime-type" : {string},
  "signed_attributes" : [{
    "type" : {string},
    "oid" : {string},
    "encoded" : {string}
  }],
  "unsigned_attributes" : [{
    "type" : {string},
    "oid" : {string},
    "encoded" : {string}
  }],
  "content_hints" : [{
    "content_type" : {
      "oid" : {string}
    },
    "content_description" : {string}
  }]
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	No disponible	Obligatorio	Tipo de firma CMS. Los valores soportados son: <ul style="list-style-type: none"> • <i>cms</i>: firma CMS según RFC 3852 • <i>ca-des-bes</i>: firma CADES-BES (Basic Electronic Signature) según ETSI TS 101 733 • <i>ca-des-epes</i>: firma CADES-EPES (Explicit Policy-based Electronic Signature) según ETSI TS 101 733
certificate	Opcional	No disponible	Opcional	Certificado del firmante codificado en Base64.
default_digest_algorithm	Opcional	No disponible	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto.
signature_target	Opcional	No disponible	Opcional	Esta propiedad es opcional y contiene información sobre la posición relativa de la firma con respecto a los datos que se firman.
signature_target.type	Opcional	No disponible	Opcional	Sólo puede tener el valor "document" (que es el valor por defecto).
signature_target.signature_packaging	Opcional	No disponible	Opcional	Posición relativa de la firma respecto a los datos que se firman. Esta propiedad es opcional y su valor puede ser "detached" o "enveloping" (attached), siendo "detached" el valor por defecto.
policy_identifier	Opcional	No disponible	Opcional	Información sobre la política de firma de acuerdo con la cual se realiza la firma y con la que dicha firma deberá ser validada. Esta propiedad es obligatoria si el tipo de firma que indica la propiedad <i>type</i> es "ca-des-epes". No puede estar presente si el tipo de firma es "cms" o "ca-des-bes".
policy_identifier	Obligatorio	No disponible	Obligatorio	OID de la política de firma.

policy_identifier.policy_hash.digest_algorithm_identifier.id	Obligatorio	No disponible	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512".
policy_identifier.policy_hash.digest_value	Obligatorio	No disponible	Obligatorio	<i>Hash</i> de la política de firma.
mime-type	Opcional	No disponible	Opcional	Tipo MIME de los datos firmados. Esta propiedad es opcional si el valor de la propiedad <i>type</i> es "cades-bes" o "cades-epes". No puede estar presente si dicho valor es "cms".
signed_attributes[]	Opcional	No disponible	Opcional	Contiene información de los atributos firmados adicionales que deben ponerse en la firma.
signed_attributes[].type	Obligatorio	No disponible	Obligatorio	Sólo puede tener el valor "custom" e indica el tipo de atributo firmado adicional.
signed_attributes[].oid	Obligatorio	No disponible	Obligatorio	OID que identifica el tipo de uno de los atributos firmados adicionales que deben ponerse en la firma.
signed_attributes[].encoded	Obligatorio	No disponible	Obligatorio	Valor de uno de los atributos firmados adicionales que deben ponerse en la firma
unsigned_attributes[]	Opcional	No disponible	Opcional	contiene información de los atributos no firmados que deben ponerse en la firma.
unsigned_attributes[].type	Obligatorio	No disponible	Obligatorio	Sólo puede tener el valor "custom" e indica el tipo de atributo no firmado que debe ponerse en la firma.
unsigned_attributes[].oid	Obligatorio	No disponible	Obligatorio	OID que identifica el tipo de uno de los atributos no firmados que deben ponerse en la firma.
unsigned_attributes[].encoded	Obligatorio	No disponible	Obligatorio	Valor de uno de los atributos no firmados que deben ponerse en la firma.
content_hints[]	Opcional	No disponible	Opcional	Información sobre el contenido que encapsulan los datos que se firman. Esta propiedad es opcional si el valor de la propiedad <i>type</i> es "cades-bes" o "cades-epes". No puede estar presente si dicho valor es "cms"

content_hints [].content_type.oid	Obligatorio	No disponible	Obligatorio	OID que identifica el tipo de contenido que encapsula los datos que se firman.
content_hints [].content_description	Obligatorio	No disponible	Obligatorio	Texto que describe el contenido que encapsulan los datos que se firman.

7.3.3.1.10. **Contrafirmas CMS/CAeS**

La propiedad *parameters* se representa en el JSON del siguiente modo:

```
{
  "type" : {string},
  "certificate": {string},
  "default_digest_algorithm": {string}
  "signature_target":
  {
    "type": "signature"
  },
  "policy_identifier": {
    "policy_id": {
      "oid": {string}
    },
    "policy_hash": {
      "digest_algorithm_identifier": {
        "id": {string}
      },
      "digest_value": {string}
    }
  },
  "signed_attributes" : [{
    "type" : {string},
    "oid" : {string},
    "encoded" : {string}
  }],
  "unsigned_attributes" : [{
    "type" : {string},
    "oid" : {string},
    "encoded" : {string}
  }]
}
```

Propiedad	Giltza	Idazki < 3.0	Idazki >= 3.0	Descripción
type	Obligatorio	No disponible	Obligatorio	<p>Tipo de firma CMS. Los valores soportados son:</p> <ul style="list-style-type: none"> • <i>cms</i>: firma CMS según RFC 3852 • <i>ca-des-bes</i>: firma CADES-BES (Basic Electronic Signature) según ETSI TS 101 733 • <i>ca-des-epes</i>: firma CADES-EPES (Explicit Policy-based Electronic Signature) según ETSI TS 101 733
certificate	Opcional	No disponible	Opcional	Certificado del firmante codificado en Base64
default_digest_algorithm	Opcional	No disponible	Opcional	Algoritmo de hash que debe utilizarse por defecto para obtener todos los hashes que deban computarse en la firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512", siendo "sha256" el valor por defecto.
signature_target.type	Obligatorio	No disponible	Obligatorio	<p>Sólo puede tener el valor "signature" y sirve para indicar que los datos que se firman corresponden a una firma que se desea contrafirmar. Esta firma tiene que estar siempre dentro de la estructura CMS/CADES que constituya el contenido (content) del recurso del que forme parte la definición de firma. Con respecto a esta estructura, solo se permiten las siguientes posibilidades (en caso contrario se acabará produciendo un error):</p> <ul style="list-style-type: none"> • Contiene una firma y ninguna contrafirma: en este caso se contrafirmará la firma • Contiene una firma y una secuencia de contrafirmas, de modo que la primera de ellas contrafirma a la firma y las siguientes contrafirman a la contrafirma precedente: en este caso se contrafirmará la última contrafirma de la secuencia

policy_identifier	Opcional	No disponible	Opcional	Información sobre la política de firma de acuerdo con la cual se realiza la firma y con la que dicha firma deberá ser validada. Esta propiedad es obligatoria si el tipo de firma que indica la propiedad <i>type</i> es "cades-epes". No puede estar presente si el tipo de firma es "cms" o "cades-bes"
policy_identifier	Obligatorio	No disponible	Obligatorio	OID de la política de firma.
policy_identifier.policy_hash.digest_algorithm_identifier.id	Obligatorio	No disponible	Obligatorio	Algoritmo para calcular el hash de la política de firma. Su valor puede ser "sha1", "sha256", "sha384" o "sha512"
policy_identifier.policy_hash.digest_value	Obligatorio	No disponible	Obligatorio	<i>Hash</i> de la política de firma.
signed_attributes[]	Opcional	No disponible	Opcional	Contiene información de los atributos firmados adicionales que deben ponerse en la firma.
signed_attributes[].type	Obligatorio	No disponible	Obligatorio	Sólo puede tener el valor "custom" e indica el tipo de atributo firmado adicional.
signed_attributes[].oid	Obligatorio	No disponible	Obligatorio	OID que identifica el tipo de uno de los atributos firmados adicionales que deben ponerse en la firma.
signed_attributes[].encoded	Obligatorio	No disponible	Obligatorio	Valor de uno de los atributos firmados adicionales que deben ponerse en la firma.
unsigned_attributes[]	Opcional	No disponible	Opcional	contiene información de los atributos no firmados que deben ponerse en la firma.
unsigned_attributes[].type	Obligatorio	No disponible	Obligatorio	Sólo puede tener el valor "custom" e indica el tipo de atributo no firmado que debe ponerse en la firma.
unsigned_attributes[].oid	Obligatorio	No disponible	Obligatorio	OID que identifica el tipo de uno de los atributos no firmados que deben ponerse en la firma.
unsigned_attributes[].encoded	Obligatorio	No disponible	Obligatorio	Valor de uno de los atributos no firmados que deben ponerse en la firma.

7.3.2. Views

Contiene información sobre las páginas que se tienen que enviar al navegador del usuario durante el proceso de firma. Esta propiedad es opcional. Si se omite, Giltza enviará al usuario una página predeterminada de reconocimiento de lectura del documento.

La estructura se ve en el siguiente JSON:

```
{
  "document_agreement" :
  {
    "skip_server_id" : {boolean},
    "document_info" :
    {
      "html_body_content" : {string}
    }
  }
}
```

Propiedad	Tipo	Descripción
views.document_agreement	Obligatorio	Información sobre la página de lectura del documento a firmar
views.document_agreement.skip_server_id	Opcional	Indicación de si podrá omitirse la página de reconocimiento de lectura en el caso de que la firma del documento se genere el servidor. El valor por defecto es
views.document_agreement.document_info	Obligatorio	Información relativa al documento sobre el que se realiza el proceso de firma que debe incluirse en la página de reconocimiento de su lectura
views.document_agreement.document_info.html_body_content	Obligatorio	Codificación en Base64 del cuerpo del documento HTML. Los tags delimitadores del cuerpo deben excluirse de la codificación en Base64 (<body>y </body>)

7.4. Firmas múltiples

La realización de firmas múltiples a través de Giltza implica la adición de bastante lógica en el RP o aplicación cliente. Mediante Giltza se obtendrán firmas unitarias, que después habrá que ir uniendo (para firmas paralelas) o modificando (para firmas en serie) a través de la lógica de negocio del RP. Este comportamiento se debe principalmente a la implementación que tiene el core de Giltza.

Por lo tanto, hay que hacer hincapié en que el peso de la construcción de firmas múltiples recaerá sobre el RP, mientras que Giltza únicamente realizará la firma solicitada mediante los diferentes JSON con los parámetros de firma.

NOTA: A partir de la v4.1.8.0 de Giltza, se introduce la posibilidad de solicitar directamente contrafirmas (firmas en serie o *countersign*) en los formatos XML/XAdES y CMS/CAdES. En ambos casos se ofrece generar una contrafirma sobre una firma o, en caso de presentar un documento con una secuencia de contrafirmas, sobre la última contrafirma.

En el formato XML/XAdES también es posible, por un lado, indicar qué firma se debe contrafirmar cuando el documento contiene varias y, por otro lado, permite indicar si la contrafirma resultante debe incluirse dentro de la firma original o no (es decir, si es *enveloped* o *detached*). Las contrafirmas *detached* son útiles porque pueden cubrir más de una firma y facilitan mantener la longevidad al margen de la firma original.

Las firmas múltiples que se pueden realizar mediante Giltza se describen a continuación.

7.4.1. PAdES en serie (countersign)

El proceso a seguir tiene los siguientes puntos:

1. Se realiza una firma PAdES, por ejemplo, enviando a Giltza un JSON similar al siguiente:

```
{
  "process_type": "urn:safelayer:eidas:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eidas:policiess:sign:document:pdf",
    "parameters" : {
      "type" : "pades-bes"
    }
  },
  "ui_locales" : <ui_locales>,
  "finish_callback_url" : <finish_callback_url>
}
```

Tener en cuenta que:

- Si es la primera firma, se le pasará a Giltza el documento PDF original.
- A partir de la segunda firma y sucesivas, habrá que pasar a Giltza el documento PDF previamente firmado.

2. Finalmente, se obtiene el documento PDF firmado.

7.4.2. XAdES Externally Detached en paralelo (cosign)

El proceso consta de los siguientes puntos:

1. Partiendo de un documento original, se realiza una firma XAdES Externally Detached, cuya estructura JSON de ejemplo puede ser:

```

{
  "process_type": "urn:safelayer:eididas:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eididas:policiess:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "raw_reference",
            "uri": "id-8767"
          }
        ]
      }
    }
  },
  "ui_locales": <ui_locales>,
  "finish_callback_url": <finish_callback_url>
}

```

2. Para realizar la firma en paralelo, se vuelve a utilizar el mismo JSON del primer punto pasándole de nuevo el documento original. El *callback* del RP se encargará de concatenar las N firmas en paralelo y añadir un nodo raíz (cuyo nombre es indiferente).
3. Por último, se obtiene un documento XML que contiene en su interior N firmas XAdES Externally Detached. El formato será como el que se muestra:

```

<raiz>
  <dsig:Signature1></dsig:Signature1>
  <dsig:Signature2></dsig:Signature2>
  <dsig:Signature3></dsig:Signature3>
  ...
  <dsig:SignatureN></dsig:SignatureN>
</raiz>

```

NOTA: La implementación descrita permite realizar tantas firmas en paralelo como se desee. Habrá que tener en cuenta que el fichero de entrada que se indique para firmar va a ser siempre el mismo.

7.4.3. XAdES Enveloped en serie I (countersign)

El proceso general engloba los siguientes puntos de interés:

1. Partiendo de un documento original, se realiza una firma XAdES Externally Detached, cuya estructura JSON de ejemplo puede ser:

```

{
  "process_type": "urn:safelayer:eididas:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eididas:polices:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "raw_reference",
            "uri": "id-8767"
          }
        ]
      }
    }
  },
  "ui_locales": <ui_locales>,
  "finish_callback_url": <finish_callback_url>
}

```

- Se añaden a la firma devuelta por Giltza los nodos XML `<xades:UnsignedProperties><xades:UnsignedSignatureProperties><xades:CounterSignature>` de manera programática mediante la clase *controller* del RP (si es que no existen previamente). Al tratarse de propiedades no firmadas, la inserción de estos nodos no influye para nada en la validación de la integridad de la firma XAdES a efectos globales.
- Se realiza una firma en serie *XAdES Internally Detached* pasando la firma obtenida en el primer punto con los nuevos nodos XML ya añadidos. Para ello, habrá que pasar a Giltza una JSON con la siguiente estructura:

```

{
  "process_type": "urn:safelayer:eididas:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eididas:polices:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "signature_target": {
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "document_reference",
            "xpath": "/dsig:Signature/dsig:SignatureValue"
          }
        ],
        "signature_placement": {
          "type": "last_child_of",
          "xpath": "/dsig:Signature/dsig:Object//xades:CounterSignature"
        }
      }
    }
  },
  "ui_locales": <ui_locales>,
  "timestamp": {
    "provider_id": "urn:safelayer:twspolices:generation:esignsp"
  },
  "finish_callback_url": <finish_callback_url>
}

```

En este punto, se comprueba el número de nodos `<dsig:Signature>` que existen dentro del nodo `<xades:CounterSignature>`:

- Si no existen, el JSON utilizado no se modifica.
- Si existen, el JSON utilizado será modificado, concretamente, el parámetro `nodes_to_sign.xpath` con el siguiente valor :

```

/dsig:Signature/dsig:Object//xades:CounterSignature/dsig:Signature[{0}]/dsig:SignatureValue

```

Donde '{0}' corresponderá al número de nodos `<dsig:Signature>` incluidos en el nodo `<xades:CounterSignature>`.

De esta forma, se asegura que la última firma en serie que se realiza haga referencia justamente a la anterior (y no siempre a la primera que se ha generado). En otras palabras, las firmas en serie sucesivas tienen que ser en "cascada".

4. Finalmente, se obtiene a través de Giltza la firma XAdES Enveloped en serie. La estructura de la firma será como la que se muestra a continuación:

```
<dsig:Signature>
...
<dsig:Object>
  <xades:QualifyingProperties>
    ...
    <xades:UnsignedProperties>
      <xades:UnsignedSignatureProperties>
        <xades:CounterSignature>
          <dsig:Signature1></dsig:Signature1>
          <dsig:Signature2></dsig:Signature2>
          ...
          <dsig:SignatureN></dsig:SignatureN>
        </xades:CounterSignature>
      </xades:UnsignedSignatureProperties>
    </xades:UnsignedProperties>
  </xades:QualifyingProperties>
</dsig:Object>
</dsig:Signature>
```

NOTA: La implementación descrita permite realizar tantas firmas en serie como se desee. Habrá que tener en cuenta los ficheros de entrada que se indican, dependiendo de si es la primera firma (punto 1) o posteriores (serie).

7.4.4. XAdES Enveloped en serie II (countersign)

Se trata de una alternativa más sencilla a la descrita en el anterior punto, sólo disponible a partir de Giltza v4.1.8.0. En este caso, la RP se libera de la mayor parte de la lógica necesaria para componer la estructura de una firma XAdES en serie.

El proceso general engloba los siguientes puntos de interés:

1. Partiendo de un documento original, se realiza una firma XAdES Externally Detached, cuya estructura JSON de ejemplo puede ser:

```

{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:policies:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "raw_reference",
            "uri": "id-8767"
          }
        ]
      }
    }
  },
  "ui_locales": <ui_locales>,
  "finish_callback_url": <finish_callback_url>
}

```

2. Sobre la firma recién obtenida, se realiza una contrafirma utilizando el siguiente JSON de ejemplo:

```

{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:policies:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "signature_target": {
        "type": "signature",
        "signature_packaging": "enveloped",
        "nodes_to_sign": [
          {
            "type": "signature_reference",
            "xpath": "/dsig:Signature"
          }
        ]
      }
    }
  },
  "ui_locales": ["es_ES"],
  "finish_callback_url": "http://localhost"
}

```

En caso de querer insertar más firmas en serie, habrá que modificar el parámetro *nodes_to_sign.xpath* hasta el nodo *<Signature>* de la contrafirma, de la siguiente manera:

```

/dsig:Signature/dsig:Object/xades:QualifyingProperties/xades:UnsignedProperties/xades:UnsignedSignatureProperties/xades:CounterSignature/dsig:Signature

```

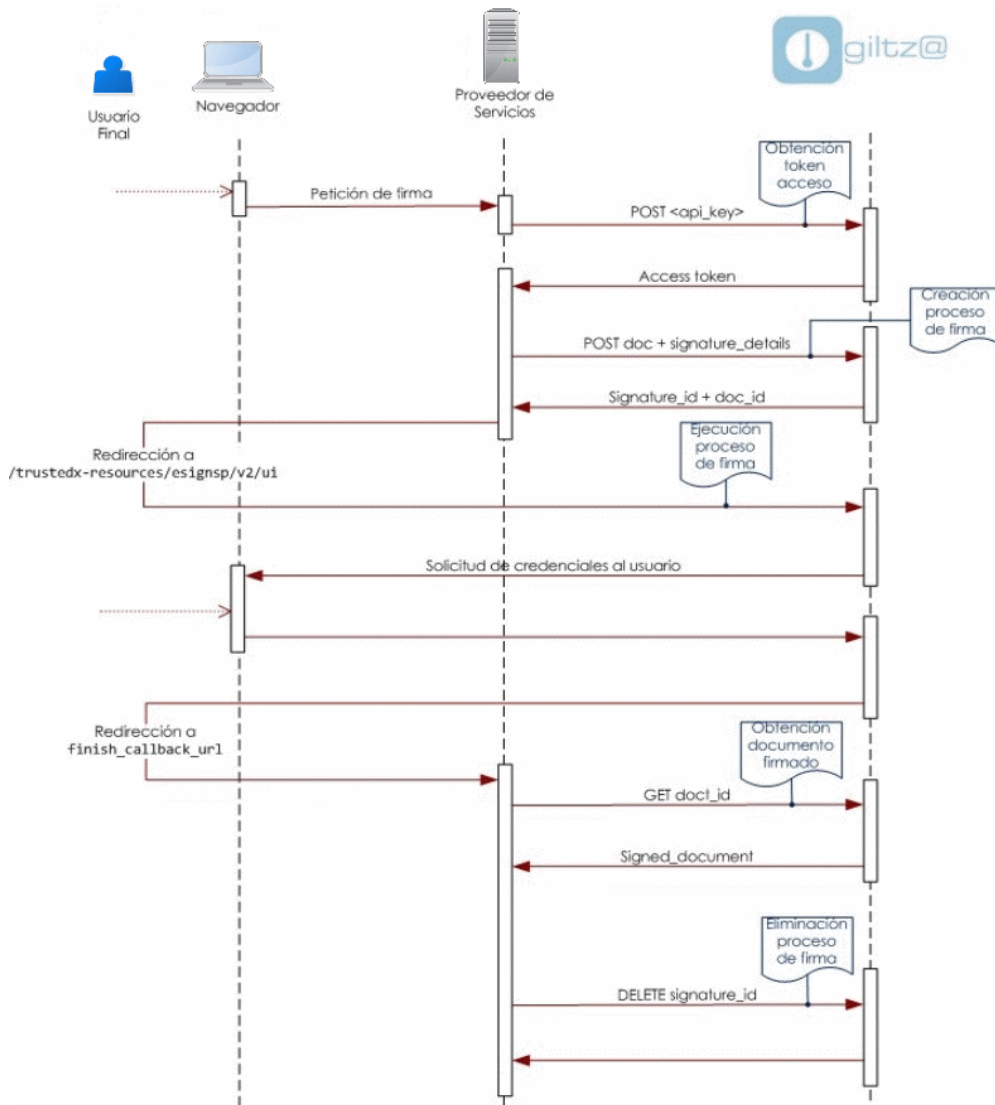
De esta forma, se asegura que la firma en serie se realiza justamente sobre la última (y no siempre sobre la primera que se ha generado). En otras palabras, las firmas en serie sucesivas se generarán en "cascada".

3. Finalmente, se obtiene a través de Giltza la firma XAdES Enveloped en serie. La estructura de la firma será como la que se muestra a continuación:

```
<dsig:Signature>
  ...
  <dsig:Object>
    <xades:QualifyingProperties>
      ...
      <xades:UnsignedProperties>
        <xades:UnsignedSignatureProperties>
          <xades:CounterSignature>
            <dsig:Signature1>
              ...
            </dsig:Signature1>
          </xades:CounterSignature>
        </xades:UnsignedSignatureProperties>
      </xades:UnsignedProperties>
    </xades:QualifyingProperties>
  </dsig:Object>
  <dsig:Object>
    <xades:QualifyingProperties>
      ...
      <xades:UnsignedProperties>
        <xades:UnsignedSignatureProperties>
          <xades:CounterSignature>
            <dsig:SignatureN></dsig:SignatureN>
          </xades:CounterSignature>
        </xades:UnsignedSignatureProperties>
      </xades:UnsignedProperties>
    </xades:QualifyingProperties>
  </dsig:Object>
  </dsig:Signature1>
  </xades:CounterSignature>
</xades:UnsignedSignatureProperties>
</xades:UnsignedProperties>
</xades:QualifyingProperties>
</dsig:Object>
</dsig:Signature>
```

7.5. Diagrama de flujo para firma

El siguiente diagrama muestra el flujo de mensajes definidos en Integración con el proveedor de servicios de firma electrónica.



8. Federación de identidades

Giltza puede federarse con otros sistemas proveedores de identidad, como redes sociales y sistemas gestionados por las administraciones o compañías privadas, de forma que se delegue en ellos la autenticación de los usuarios, total o parcialmente. La aceptación de identidades externas contribuye a reducir el volumen de credenciales que deben manejar los usuarios y, por lo tanto, facilita el uso de las aplicaciones proveedoras de servicios.

En este punto, se va a describir la manera de federarse en Giltza Profesional a través de Giltza (ámbito ciudadanía).

8.1. Configuración

Para poder federar la identidad a Giltza Profesional a través de Giltza y explotar la información, será necesaria la siguiente configuración:

Propiedad	Valor
acr_values	urn:safelayer:twspolicies:authentication:flow:giltza:profesional
scope	urn:safelayer:eidas:external_info

El flujo indicado en la propiedad *acr_values* sólo se mostrará si se indica explícitamente en el inicio de la autorización OAuth 2.0 de la RP. En caso contrario, no será visible para el usuario. Para concatenar varios valores en esta propiedad, hay que utilizar el carácter "|".

Una vez realizada la autenticación federada, en el JSON correspondiente al UserInfo se incluirá un atributo denominado *external_info*, que incluye toda la información de la autenticación proporcionada por Giltza Profesional (IdP federado).

NOTA: Si después de la autenticación, se desea realizar algún proceso adicional sobre las APIs de Giltza (firma de documentos, obtención de información, etc.) con el usuario en curso, habrá que tener en cuenta que el host al que hay que apuntar tiene que ser el de Giltza Profesional (ya que es el IdP sobre el que se ha realizado la autenticación).

9. Status de las respuestas HTTP

El código de estado (status) de la respuesta HTTP indica el resultado de la operación, según la convención descrita en la RFC 7231. En caso de error, el formato de la respuesta puede variar según el tipo de error.

9.1. Códigos de estado de operaciones completadas correctamente

Cuando una operación se puede completar de forma satisfactoria, es posible que el cuerpo de la respuesta contenga más información sobre el resultado de la operación.

Código HTTP	Descripción
200 OK	La operación se ha completado correctamente. El cuerpo de la respuesta puede incluir más información sobre el resultado de la operación
201 Created	La operación de creación de un nuevo recurso se ha completado correctamente. El cuerpo de la respuesta incluye información sobre el recurso creado o la cabecera <i>Location</i> contiene la URL de acceso a éste
204 No content	La operación se ha completado correctamente. No se obtiene ninguna entidad como respuesta y el cuerpo está vacío

9.2. Códigos de estado con parámetro error en la cabecera HTTP WWW-Authenticate

Cuando se produce un error de autorización, se obtiene un código de error en el parámetro *error* de la cabecera HTTP *WWW-Authenticate* de la respuesta, como se describe en RFC 6750. Por ejemplo:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer error="invalid_token"
```

A continuación, se listan los códigos de estado que pueden ir acompañados de una cabecera *WWW-Authenticate*, y los posibles valores del parámetro *error* en cada caso.

Código HTTP	Descripción
400 Bad Request	<i>invalid_request</i> : no es posible procesar la petición debido a una sintaxis incorrecta de la cabecera HTTP <i>Authorization</i> , posiblemente porque está mal formada o hay más de una

401 Unauthorized	<p>Se pueden dar los siguientes casos:</p> <ul style="list-style-type: none"> • <i>Sin código de error</i>: las condiciones de autenticación requeridas no han sido satisfechas porque la petición no incluye una cabecera <i>HTTP Authorization</i>, o bien la incluye, pero con un esquema de autenticación HTTP no reconocido • <i>invalid_token</i>: el token de autorización no es válido o está caducado. En el caso de los tokens OAuth, el token caduca al cabo del número de segundos indicado en el resultado de la operación Obtener un token de acceso (parámetro <i>expires_in</i>)
403 Forbidden	<p><i>insufficient_scope</i>: los scopes a los que está asociado el token de acceso OAuth no son suficientes para acceder al recurso protegido. La cabecera <i>WWW-Authenticate</i> incluye un parámetro adicional <i>scope</i> con el scope esperado. Un token OAuth está asociado a los scopes que se hayan indicado en la operación Obtener autorización u Obtener un token de acceso, según el tipo de grant OAuth que se haya utilizado para obtener el token</p>

9.3. Códigos de estado con descripción opcional del error en el cuerpo de la respuesta

Para cualquier otro tipo de error, es habitual que el cuerpo de la respuesta contenga una estructura de datos JSON con los siguientes parámetros:

- *error*: código de error
- *error_description*: descripción adicional del error (opcional)
- *error_details*: detalles adicionales del error. El formato es un objeto JSON con subcampos (opcional)

Por ejemplo:

```
HTTP/1.1 400 Bad Request
{
  "error": "NotSupportedException",
  "error_description": "Cannot consume content type",
  "error_details": { }
}
```

A continuación, se describen posibles valores del parámetro *error* para varios códigos de estado.

Código HTTP	Descripción
400 Bad Request	<p>La sintaxis de la petición no es correcta o no se cumplen las condiciones requeridas para completar la operación:</p> <ul style="list-style-type: none"> • <i>AccountAlreadyExistsException</i>: el valor del atributo único indicado en la petición y que se desea usar como identificador explícito de la cuenta de usuario ya está asignado a otra cuenta • <i>AccountAttributeNotDefinedException</i>: alguno de los atributos indicados en la petición no está definido en el dominio de identidad o no está mapeado a la cuenta de usuario • <i>AmbiguousSignIdentityTypeException</i> y <i>MultipleIdentitiesSelectedException</i>: las etiquetas solicitadas al proveedor de firma para crear una firma digital identifican a más de una identidad de firma disponible. • <i>BadRequestException</i>: la petición no es correcta • <i>BlockedDeviceException</i>: el dispositivo de firma para el que se solicita la operación tiene un proceso pendiente de completarse • <i>DeviceAlreadyExistsException</i>: el dispositivo de firma que se solicita registrar ya existe • <i>DeviceTypesNotExistsException</i>: el tipo de dispositivo de firma indicada en la creación de la identidad de firma no se corresponde con ninguno registrado en la configuración • <i>InconsistentLinkedAccountIdException</i>: se ha intentado actualizar la información de una cuenta vinculada indicando un identificador de dominio externo (campo <i>domain</i>) o un identificador de la identidad externa (campo <i>account_id</i>) que no coinciden con los que ya estaban asociados a la identidad vinculada. Una vez creada la identidad vinculada, estas propiedades no pueden modificarse • <i>IncorrectSignIdentitySelectorsException</i>: el selector de identidades de firma solicitado es incorrecto o no existe ninguna identidad asociada • <i>InvalidAttributeValueException</i>: alguno de los atributos indicados en la petición no tiene valor o tiene un formato no soportado • <i>InvalidExternalDomainNameException</i>: el nombre del dominio externo indicado en el cuerpo de la petición no es válido o el dominio externo no está federado al dominio al que pertenece la cuenta de usuario

- *InvalidCsrTemplateException*: la plantilla para generar la petición de certificación indicada en la creación de la identidad de firma solicitada es incorrecta
- *InvalidKeyTemplateException*: la plantilla para generar las claves indicada en la creación de la identidad de firma solicitada es incorrecta
- *InvalidParametersException*: el valor de uno o más parámetros indicados en la petición no se soporta
- *InvalidPkcs12Exception*: el PKCS#12 y la contraseña recibidos para crear la identidad son incorrectos o no se pueden emplear
- *InvalidQueryFilterException*: el parámetro *filter* indicado en la petición está mal formado, incluye un atributo que no está definido en el dominio de identidad o no está mapeado a las cuentas de usuario, o el atributo no está definido como único
- *InvalidRequestException*: la sintaxis de la petición es incorrecta o el valor de algunos parámetros no están soportados
- *JsonParseException* y *JsonProcessingException*: el cuerpo de la petición está malformado
- *LinkedAccountAlreadyExistsException*: la cuenta de usuario ya tiene una identidad vinculada con el dominio externo (campo *domain*) y el identificador de la identidad externa (campo *account_id*) especificados
- *MandatorySignatureException*: el proceso de firma no tiene definida una definición de firma
- *ModelValidationException*: la petición no es correcta porque falta algún parámetro obligatorio en la URL o en el cuerpo, o porque el contenido del cuerpo está vacío o no es completo
- *MissingExternalAccountIdException*: el cuerpo de la petición no incluye el identificador de la identidad externa en el dominio de identidad federado (campo *account_id*)
- *MissingUserIdAttributeException*: el cuerpo de la petición no incluye el valor del atributo que se usa para determinar el identificador explícito de la cuenta de usuario

	<ul style="list-style-type: none"> • <i>MultipleAccountsForUniqueAttributeException</i>: el valor de un atributo único indicado en la petición ya está asignado a otra cuenta de usuario • <i>MultipleValuesForUniqueAttributeException</i>: un atributo único indicado en la petición tiene múltiples valores • <i>NotSupportedException</i>: la cabecera <i>Content-type</i> de la petición no es la esperada. La mayoría de las operaciones esperan el valor <i>application/json</i> en esta cabecera • <i>NotSupportedSignatureParametersException</i> y <i>UnexpectedSignerParametersTypeException</i>: los parámetros solicitados en la definición de firma son incorrectos • <i>SignaturePolicyNotFoundException</i>: no se ha indicado ninguna política de firma • <i>SignaturePolicyNotSupportedException</i>: la política de firma solicitada no está soportada • <i>UndefinedIdentityDomainException</i>: el token de autorización está asociado a un dominio de identidad no definido • <i>UnmatchingSignIdentityTypeException</i> y <i>UnsuitableDeviceForSignIdentityException</i>: las etiquetas de la identidad de firma que se solicita crear no tienen correspondencia con ningún tipo de identidad establecido en la configuración • <i>UnsupportedSignIdentityCapabilityException</i>: la identidad de firma especificada no es compatible con la operación solicitada
401 Unauthorized	<p>La petición no incluye credenciales de autenticación válidas:</p> <ul style="list-style-type: none"> • <i>UserAuthenticationRequiredException</i>: el token de acceso no está asociado a un usuario

403 Forbidden

El servidor no autoriza el acceso al recurso, por ejemplo, porque el tipo de autorización utilizada no es correcta o no ofrece permisos suficientes:

- *DisabledSignIdentityException*: la identidad de firma está inhabilitada.
- *InsufficientPermissionsException* y *PermissionsRequiredException*: los permisos otorgados por el token de autorización no son suficientes para acceder al recurso protegido. Si se ha utilizado un token JWT, los permisos del token son fijos. Vea Autorización para utilizar las APIs para más información.
- *InsufficientPermissionsForAttributeException*: los permisos otorgados por el token de autorización no son suficientes para obtener alguno de los atributos solicitados en la petición.
- *InvalidPrivilegeException* y *NoSuchPrivilegeException*: los permisos otorgados por el token de autorización no son suficientes para ejecutar la operación.
- *NotAuthorizedException*: el token de autorización no se ha obtenido con el flujo OAuth adecuado, o bien no se ha especificado ni se puede deducir el dominio de identidad.
- *LockedSignIdentityException*: la identidad de firma en servidor está bloqueada.
- *UserNotAuthorizedException*: el token de autorización no debería estar asociado a un usuario, o no está asociado al mismo usuario que se indica en la petición.
- *UserDomainAccessRequiredException*: se requiere autorización administrativa de acceso al dominio de identidad. Si se ha autorizado la llamada mediante un *token* OAuth, este error puede ser debido a que el *token* ha sido emitido por un servidor de autorización que no está asociado a ningún dominio de identidad.

404 Not Found	<p>Recurso no encontrado:</p> <ul style="list-style-type: none"> • <i>DocumentNotFoundException</i> y <i>DocumentNotFoundExceptionRuntime</i>: el documento sobre el que se desea realizar una operación no existe • <i>ItemNotFoundException</i> y <i>NotFoundException</i>: el recurso sobre el que se desea realizar una operación no existe • <i>PasswordNotSetException</i>: no existe una contraseña registrada con ese nombre para el usuario • <i>ProcessNotFoundException</i>: el proceso de firma de un documento sobre el que se desea realizar una operación no existe • <i>UnknownLinkedAccountException</i>: la identidad vinculada indicada en la petición no pertenece al usuario • <i>UnknownUserAccountException</i>: no existe la cuenta de usuario indicada en la petición
405 Method Not Allowed	<p>El método HTTP indicado en la petición no es adecuado para acceder al recurso.</p> <ul style="list-style-type: none"> • <i>NotAllowedException</i>: el servicio no soporta la operación HTTP indicada en la petición
409 Conflict	<p>La modificación solicitada en la petición no puede aplicarse dado el estado actual del recurso:</p> <ul style="list-style-type: none"> • <i>InvalidStateException</i>: el estado del recurso no permite realizar la operación
410 Gone	<p>El recurso no existe ni existirá</p>
415 Unsupported	<p>Los datos enviados al servidor no son del tipo requerido por la operación</p>
422 Unprocessable Entity	<p>La modificación solicitada es correcta, pero no puede realizarse porque dejaría al recurso en un estado inválido:</p> <ul style="list-style-type: none"> • <i>InconsistentUpdateRequestException</i>: no se pueden aplicar simultáneamente los cambios en el recurso
500 Internal Server Error	<p>Error no esperado por el servidor. En estos casos, el administrador de Giltza deberá diagnosticar el problema</p>

10. Ejemplos de integración mediante cURL

cURL es un cliente de HTTP que permite de una manera muy sencilla construir mensajes y enviar peticiones REST a un servidor.

cURL también dispone de librerías y enlaces con múltiples lenguajes de programación (<http://curl.haxx.se/libcurl/bindings.html>), viene pre-instalado en la mayoría de los sistemas Linux y se puede descargar para Windows u otros sistemas en curl.haxx.se/.

10.1. Servicio de autenticación

En este punto se describe como invocar los servicios de autenticación usando la versión de línea de comandos de cURL.

- Obtención de la autorización.
- Obtención del token de acceso.
- Consulta de los datos de usuario.

10.1.1. Obtención de la autorización

La petición de autorización desencadena el proceso de autenticación que requiere de interacción con el usuario, por lo tanto, se va a utilizar cualquier navegador para copiar la siguiente URL e iniciar el flujo:

```
https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe?client_id=<client_id>&response_type=code&
redirect_uri=https%3A%2F%2Flocalhost%2Fauthentication%
2Fcallback&state=123456&acr_values=urn%3Asafelayer%3Atws%3Apolicias%
3Aauthentication%3Aflow%3Acert&scope=profile%20email
```

Interaccionar con el proceso de autenticación desde el navegador y, si la autenticación acaba con éxito, en la URL del navegador debería aparecer lo siguiente:

```
https://localhost/authentication/callback?code=
697839fc2fc335f77fec5d1aded93ebe21c3111a45a658b0568cf2609cf8ff2b&state=123456
```

10.1.2. Obtención del token de acceso

Copiar el valor del parámetro code (en rojo) en el siguiente comando cURL que invoca una petición de token:

```
cURL https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe/token -i -k -H
"Authorization: Basic <api_key>" --data
"grant_type=authorization_code&redirect_uri=https%3A%2F%2Flocalhost%
2Fauthentication%2Fcallback&code=<code>"
```

Si la petición de token se ejecuta correctamente, el cuerpo de la respuesta debe ser parecida a la siguiente:

```
{"scope":"email profile","expires_in":120,"token_type":"bearer","access_token":
"3dbd84c4e2256182ff31d183ec8786e3cb7cc47ef318a67cde8d7d759659d391"}
```

10.1.3. Consulta de los datos de usuario

Copiar el valor del parámetro access_token (en azul) en el siguiente comando cURL que invoca una petición de userInfo:

```
cURL https://eidas.izenpe.com/trustedx-resources/openid/v1/users/me -i -k -H "Authorization:
Bearer <access_token>"
```

Y, si todo va correctamente, el cuerpo de la respuesta debe ser parecido a lo siguiente:

```
{"sub":"11111111A","domain":"izenpe","acr":"urn:safelayer:twspolicies:authentication:flow:cert",
"amr":["urn:ietf:rfc:2246"]}
```

10.2. Servicio de firma

En este punto se describe como invocar los servicios de firma usando la versión de línea de comandos de cURL.

- Obtención del tokenOAuth
- Creación del proceso defirma
- Ejecución del proceso defirma
- Obtención del documentofirmado
- Eliminación del proceso defirma

10.2.1. Obtención del token OAuth

Antes de comenzar con el proceso de firma, es necesario obtener el token de acceso que no habilitará para consumir los recursos de firma. Con este fin, introducimos el siguiente comando:

```
curl -X POST "https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/token" -H "Content-Type: application/x-www-form-urlencoded" -H "Authorization: Basic <api_key>" -d 'grant_type=client_credentials&scope=urn%3Asafelayer%3Aeidasp%3Aprocess%3Adocument'
```

Si la petición de token se ejecuta correctamente, el cuerpo de la respuesta debe ser parecida a la siguiente:

```
{"scope":"urn:safelayer:eidas:sign:process:document","expires_in":600,"token_type":"Bearer","access_token":"ccc6132adeeac477f039bcf9b3ae8b5a7f67abf5c20b63364e8a76034693b78c"}
```

10.2.2. Creación del proceso de firma

Esta petición de servicio crea e inicia el proceso de firma. El comando cURL correspondiente es:

```
curl -X POST "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes" -H "Authorization: Bearer ccc6132adeeac477f039bcf9b3ae8b5a7f67abf5c20b63364e8a76034693b78c" -F "process=@process.json" -F "document=@doc.pdf"
```

El valor "*doc.pdf*" es el nombre del fichero PDF que se enviará a firmar y "*process.json*" es el fichero con las características de la firma que se va a realizar. Ambos deben existir en el mismo directorio en el que se ejecuta el comando cURL. El valor en azul corresponde con el token de acceso obtenido en la respuesta del mensaje anterior.

El contenido mínimo del fichero "*process.json*" es:

```
{
  "process_type" : "urn:safelayer:eidas:processes:document:sign:esigp",
  "signer" : {
    "signature_policy_id": "urn:safelayer:eidas:policiess:sign:document:pdf"
  },
  "labels": [{"bak"}],
  "finish_callback_url" : "http://localhost"
}
```

Donde indicamos que la firma se realizará con una identidad de tipo Bak y la que URL de redirección es "*http://localhost*".

Si el proceso de firma se crea e inicia correctamente se retornará una respuesta en formato JSON parecida a la siguiente:

```
{"process_type":"urn:safelayer:eidas:processes:sign:esigp","id":"scs8la1ugnn97qc2bmdot3p5dcd7qdii","self":"https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/scs8la1ugnn97qc2bmdot3p5dcd7qdii","tasks":{"pending":[{"type":"UserBrowserTask","id":"nqg3s8rpakc8ht1l7jn1172d538pmco5","url":"https://eidas.izenpe.com/trustedx-resources/esignsp/v2/ui?"}
```

```
signerProcessId=scs8la1ugnn97qc2bmdot3p5dcu7qdii"}}, "documents":  
[{"url": "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/  
gukt0107hqnp0qhl9q5737th58igj9"}]}
```

10.2.3. Ejecución del proceso de firma

La ejecución del proceso de firma implica interacción con el usuario, por lo tanto, deberá llevarse a cabo dentro de un navegador. Para ello, simplemente abrir un navegador y copiar y pegar la URL señalada en rojo en la respuesta anterior, y seguir el flujo de ejecución como usuario.

Si el proceso de firma se ejecuta correctamente, entonces en la URL del navegador debería aparecer la dirección que se ha utilizado en la redirección de retorno, en este caso concreto *"http://localhost"* (definida en el fichero *"process.json"*), y debería verse en el navegador de forma parecida a lo siguiente:

```
http://localhost?status=finished
```

10.2.4. Obtención del documento firmado

Para recuperar el documento firmado ejecutar el siguiente comando cURL copiando el identificador del documento de la respuesta del mensaje de creación de la firma (en negrita):

```
curl -X GET "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/documents/  
gukt0107hqnp0qhl9q5737th58igj9/content" -H "Authorization: Bearer  
ccc6132adeeac477f039bcf9b3ae8b5a7f67abf5c20b63364e8a76034693b78c"
```

Y el resultado será la descarga en formato binario del documento firmado con nombre *"doc-signed.pdf"*, el cual se puede abrir y comprobar la firma realizada.

10.2.5. Eliminación del proceso de firma

Por último, para eliminar el proceso de firma realizado, se ejecutará el siguiente comando:

```
curl -X DELETE "https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/  
scs8la1ugnn97qc2bmdot3p5dcu7qdii" -H "Authorization: Bearer  
ccc6132adeeac477f039bcf9b3ae8b5a7f67abf5c20b63364e8a76034693b78c"
```

11. Ejemplos de integración mediante Java

Con el objetivo de facilitar al máximo la integración de los diferentes aplicativos (RPs) con Giltza, en este apartado se van a describir ejemplos en código Java de las operaciones más comunes realizadas sobre las diferentes APIs disponibles.

En los siguientes puntos se van a describir aquellas partes de código Java más relevantes para la integración con Giltza. Para un mayor nivel de detalle, **Izenpe** dispone de ejemplos ya preparados para realizar llamadas a las diferentes operaciones ofrecidas por las APIs de Giltza.

11.1. Servicio de autenticación

Para iniciar un flujo de autorización OAuth 2.0, los pasos a realizar en el RP son los siguientes:

1. Se genera un valor aleatorio de 16 bytes codificado en Base64, para mantener el estado entre la petición OAuth de inicio y el *callback* OAuth. Giltza mantendrá este valor exacto cuando redirija el navegador de vuelta a la aplicación. Este estado se utiliza para prevenir ataques CSRF, tal y como recomienda la especificación.

```
SecureRandom random = new SecureRandom();
byte bytes[] = new byte[16];
random.nextBytes(bytes);
String state = Base64.encodeBase64String(bytes);
```

2. Se construye la URL de la petición para iniciar el flujo OAuth2.0, de la siguiente forma.

```
String oauthUri = newURIBuilder("https://eididas.izenpe.com/trustedx-authserver/oauth/izenpe")
    .setParameter("redirect_uri", "http://localhost:8081/minimalist-oauth-rp/oauthCallback")
    .setParameter("client_id", "appOAuthClientId")
    .setParameter("response_type", "code")
    .setParameter("state", state)
    .setParameter("scope", "urn:safelayer:eididas:authn_detailsurn:izenpe:identity:global")
    .setParameter("acr_values", "urn:safelayer:tw:policias:authentication:level:low")
    .setParameter("ui_locales", "es").build().toASCIIString();
```

3. Se inicia el flujo OAuth 2.0 redireccionando el navegador al *endpoint* OAuth de Giltza.

```
response.sendRedirect(oauthUri);
```

4. Una vez que la autenticación en Giltza se ha realizado con éxito, a través del *callback* del RP, se va a solicitar un token de acceso mediante una petición HTTP POST.

```

HttpPost httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-authserver/oauth/izenpe/
token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " +apiKey);

// El resto de parametros se envían en el cuerpo de la petición
ArrayList<NameValuePair> postParameters;
postParameters = new ArrayList<NameValuePair>();
postParameters.add(new BasicNameValuePair("redirect_uri", "http://localhost:8081/minimalist-
oauth-rp/oauthCallback"));
postParameters.add(new BasicNameValuePair("code", code));
postParameters.add(new BasicNameValuePair("grant_type", "authorization_code"));
httpPost.setEntity(new UrlEncodedFormEntity(postParameters));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse =httpClient.execute(httpPost);

// Se lee la respuesta de Giltza
HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONTokener(tokenResponse));

// Se obtiene el token de acceso
String accessToken = (String) tokenJson.get("access_token");

```

5. Se recuperan los atributos del usuario autenticado mediante una petición HTTP GET.

```

HttpGet httpGet = new HttpGet("https://eidas.izenpe.com/trustedx-resources/openid/v1/users/me");

// Se envía el token de acceso utilizando la cabecera WWW-Authentication
httpGet.setHeader("Authorization", "Bearer " + accessToken);
httpResponse =httpClient.execute(httpGet);
reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-
8"));

// userInfoResponse contiene una estructura JSON
String userInfoResponse = reader.readLine();
JSONObject userInfoJson = new JSONObject(new JSONTokener(userInfoResponse));

// Se recuperan los parametros JSON que se crean oportunos
String subject = (String) userInfoJson.get("sub");

```

6. Finalmente, se redirige al usuario recién registrado, a la página principal del aplicativo (por ejemplo).

```

response.sendRedirect("http://localhost:8081/minimalist-oauth-rp/home");

```

11.2. Servicio de firma

Para iniciar un proceso de firma, los pasos a realizar en el RP son los siguientes, dependiendo del tipo de operación que se desee solicitar.

11.2.1. Firma PAdES

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un *grant* del tipo *client_credentials*.

```

HttpPost httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/
token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " +apiKey);

// El resto de parametros se envian en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type","client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eidas:sign:process:document");
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse =httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(),"UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(newJSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");

```

2. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```

httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/
signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso
de firma
MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
multipartBuilder.setMode(HttpMultipartMode.RFC6532);
// jsonString contiene un JSON con los parametros de la firma
multipartBuilder.addTextBody("process", jsonString,ContentType.APPLICATION_JSON);
// pdfByteArray es el documento que se va a enviar a firmar a Gilta
multipartBuilder.addBinaryBody("document", pdfByteArray, ContentType.create("application/pdf",
Consts.UTF_8), pdfFileName);
httpPost.setEntity(multipartBuilder.build());

// Se envia la petición a Gilta
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-
8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

// URLsignatureProcess contiene la URL para eliminar la petición del document en Gilta
// urlUserRedirectAuthorization contiene el id del proceso defirma
// URLignedDocument contiene la URL donde se ubica el documento firmado
String URLsignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");
String URLignedDocument =
objJson.getJSONArray("documents").getJSONObject(0).get("url").toString() +"/content";

```

3. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLignatureProcess", URLignatureProcess);
request.getSession().setAttribute("URLignedDocument", URLignedDocument);
response.sendRedirect(urlUserRedirectAuthorization);
```

4. En el callback, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar el documento firmado haciendo uso de la URL indicada en la variable de sesión *URLignedDocument*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignedDocument")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " + applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

    // Si no hay ningun error, se obtiene una respuesta "application/pdf" con el documento
    firmado.
    // Se copia este documento firmado a un fichero enlocal.
    OutputStream pdfOutputStream = new FileOutputStream(request.getSession().getServletContext()
        .getRealPath("/") + "/pdf/signed.pdf");
    IOUtils.copy(httpResponse.getEntity().getContent(), pdfOutputStream);

    xmlOutputStream.close();
}
```

5. Se envía una petición HTTP DELETE a Giltza para eliminar el documento del servidor, utilizando el atributo de sesión *URLignatureProcess*.

```
HttpDelete httpDelete = new HttpDelete(request.getSession().getAttribute("URLignatureProcess")
    .toString());
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

6. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri", "http://localhost:8081/minimalist-oauth-rp/home");
// Atributo para poder descargar el fichero desde el JSP
request.setAttribute("pdfFileName", "signed.pdf");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_signature_end.jsp").forward(request,
    response);
```

<jsonString>

Se trata de la variable que contiene el JSON con los parámetros de la firma que se va a enviar a Giltza para la realización de la firma.

```

{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:policies:sign:document:pdf",
    "parameters": {
      "type": "pades-bes",
      "default_digest_algorithm": "sha256",
      "estimated_signature_size": 65000,
      "location": "Bilbao, Spain",
      "reason": "Razon de la firma",
      "signature_field": {
        "name": "user_signature",
        "location": {
          "page": {
            "number": "last"
          },
          "rectangle": {
            "x": 100,
            "y": 110,
            "height": 150,
            "width": 400
          }
        },
        "appearance": {
          "signature_details": {
            "details": [
              {
                "type": "subject",
                "title": "Signer Distinguished Name: "
              },
              {
                "type": "location",
                "title": "Signature Location: "
              },
              {
                "type": "date",
                "title": "Signature date: "
              }
            ]
          }
        }
      }
    }
  },
  "ui_locales": ["es_ES"],
  "timestamp": {
    "provider_id": "urn:safelayer:tw:policies:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}

```

11.2.2. Firma XAdES Enveloped

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un *grant* del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parametros se envian en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eidas:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");
```

2. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " + applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso de firma
MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
multipartBuilder.setMode(HttpMultipartMode.RFC6532);
// jsonString contiene un JSON con los parametros de la firma
multipartBuilder.addTextBody("process", jsonString, ContentType.APPLICATION_JSON);
// xmlByteArray es el documento que se va a enviar a firmar a Gilta
multipartBuilder.addBinaryBody("document", xmlByteArray, ContentType.create("text/xml", Consts.UTF_8), xmlFileName);
httpPost.setEntity(multipartBuilder.build());

// Se envia la petición a Gilta
httpResponse = httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(new JSONTokener(tokenResponse));

// URLSignatureProcess contiene la URL para eliminar la petición del document en Gilta
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocument contiene la URL donde se ubica el documento firmado
String URLSignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");
String URLignedDocument =
objJson.getJSONArray("documents").getJSONObject(0).get("url").toString() + "/content";
```

3. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLignatureProcess", URLignatureProcess);
request.getSession().setAttribute("URLignedDocument", URLignedDocument);
response.sendRedirect(urlUserRedirectAuthorization);
```

4. En el callback, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar el documento firmado haciendo uso de la URL indicada en la variable de sesión *URLignedDocument*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignedDocument")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " + applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

    // Si no hay ningun error, se obtiene una respuesta "text/xml" con el documento firmado. Se
    // copia este documento firmado a un fichero enlocal
    OutputStream xmlOutputStream = new FileOutputStream(request.getSession().getServletContext()
        .getRealPath("/") + "/xml/signed.xsig");
    IOUtils.copy(httpResponse.getEntity().getContent(), xmlOutputStream);

    xmlOutputStream.close();
}
```

5. Se envía una petición HTTP DELETE a Giltza para eliminar el documento del servidor, utilizando el atributo de sesión *URLignatureProcess*.

```
HttpDelete httpDelete = new HttpDelete(request.getSession().getAttribute("URLignatureProcess")
    .toString());
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

6. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri", "http://localhost:8081/minimalist-oauth-rp/home");
// Atributo para poder descargar el fichero desde el JSP
request.setAttribute("xmlFileName", "signed.xsig");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_xades_signature_end.jsp").forward(request,
response);
```

<jsonString>

Se trata de la variable que contiene el JSON con los parámetros de la firma que se va a enviar a Giltza para la realización de la firma.

```

{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:policies:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "enveloped",
        "nodes_to_sign": [
          {
            "type": "document_reference",
            "xpath": "/"
          }
        ],
        "signature_placement": {
          "type": "last_child_of",
          "xpath": "/*"
        }
      }
    }
  },
  "ui_locales": ["es_ES"],
  "timestamp": {
    "provider_id": "urn:safelayer:tw:policies:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}

```

11.2.3. Firma XAdES Externally Detached EPES

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un grant del tipo *client_credentials*.

```

HttpPost httpPost = new HttpPost("https://eid.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parámetros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eid:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");

```

2. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parámetros tienen que pasarse en el cuerpo de la petición para iniciar el proceso de firma
MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
multipartBuilder.setMode(HttpMultipartMode.RFC6532);
// jsonString contiene un JSON con los parámetros de la firma
multipartBuilder.addTextBody("process", jsonString, ContentType.APPLICATION_JSON);
// xmlByteArray es el documento que se va a enviar a firmar a Giltza
multipartBuilder.addBinaryBody("document", xmlByteArray, ContentType.create("text/xml",
Consts.UTF_8), xmlFileName);
httpPost.setEntity(multipartBuilder.build());

// Se envía la petición a Giltza
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

// URLSignatureProcess contiene la URL para eliminar la petición del documento en Giltza
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocument contiene la URL donde se ubica el documento firmado
String URLSignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");
String URLignedDocument =
objJson.getJSONArray("documents").getJSONObject(0).get("url").toString() +"/content";
```

3. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLSignatureProcess", URLSignatureProcess);
request.getSession().setAttribute("URLignedDocument", URLignedDocument);
response.sendRedirect(urlUserRedirectAuthorization);
```

4. En el *callback*, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar el documento firmado haciendo uso de la URL indicada en la variable de sesión *URLignedDocument*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignedDocument")
.toString());

    httpGet.setHeader("Authorization", "Bearer " +applicationAccesssToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse =httpClient.execute(httpGet);

    // Si no hay ningún error, se obtiene una respuesta "text/xml" con el documento firmado. Se
    // copia este documento firmado a un fichero en local
    OutputStream xmlOutputStream = new FileOutputStream(request.getSession().getServletContext()
.getPath("/") + "/xml/signed.xsig");
    IOUtils.copy(httpResponse.getEntity().getContent(), xmlOutputStream);

    xmlOutputStream.close();
}
```

5. Se envía una petición HTTP DELETE a Giltza para eliminar el documento del servidor, utilizando el atributo de sesión *URLignatureProcess*.

```
HttpDelete httpDelete = new  HttpDelete(request.getSession().getAttribute("URLignatureProcess")
.toString());
// Atributo para poder descargar el fichero desde el JSP
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

6. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri","http://localhost:8081/minimalist-oauth-rp/home");
// Atributo para poder descargar el fichero desde el JSP
request.setAttribute("xmlFileName", "signed.xsig");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_xades_signature_end.jsp").forward(request,
response);
```

<jsonString>

Se trata de la variable que contiene el JSON con los parámetros de la firma que se va a enviar a Giltza para la realización de la firma.

```
{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:policies:sign:document:xml",
    "parameters": {
      "type": "xades-epes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "raw_reference"
          }
        ]
      },
      "policy_identifier": {
        "policy_id": {
          "identifier": {
            "uri": "https://euskadi.eus/bopv2/datos/2012/07/1203474a.pdf"
          },
          "description": "Política de firma izenpe 1.5"
        },
        "policy_hash": {
          "digest_algorithm_identifier": {
            "id": "sha1"
          },
          "digest_value": "Ohix16upD6av8N7pEvDABhEL6hM="
        },
        "policy_qualifiers": [
          {
            "type": "spuri",
            "uri": "https://euskadi.eus/bopv2/datos/2012/07/1203474a.pdf"
          }
        ]
      }
    }
  },
  "ui_locales": ["es_ES"],
  "finish_callback_url": "http://localhost"
}
```

11.2.4. Firma XAdES Externally Detached a partir de un hash

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un *grant* del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " +apiKey);

// El resto de parametros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type","client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eidas:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse =httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(),"UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(newJSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");
```

2. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso de firma
httpPost.setHeader("Content-Type",ContentType.APPLICATION_JSON.getMimeType());
// jsonString contiene un JSON con los parametros de la firma, entre los que se incluye el hash a firmar
HttpEntity entityString = new StringEntity(jsonString);
httpPost.setEntity(entityString);

// Se envía la petición a Gilta
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(),"UTF-8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

// URLsignatureProcess contiene la URL para eliminar la petición del document en Gilta
// urlUserRedirectAuthorization contiene el id del proceso defirma
// URLignedDocument contiene la URL donde se ubica el documento firmado
String URLsignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");
String URLignedDocument =
objJson.getJSONArray("documents").getJSONObject(0).get("url").toString() +"/content";
```

3. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLignatureProcess", URLignatureProcess);
request.getSession().setAttribute("URLignedDocument", URLignedDocument);
response.sendRedirect(urlUserRedirectAuthorization);
```

4. En el callback, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar el documento firmado haciendo uso de la URL indicada en la variable de sesión *URLignedDocument*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignedDocument")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " + applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

    // Si no hay ningun error, se obtiene una respuesta "text/xml" con el documento firmado. Se
    // copia este documento firmado a un fichero enlocal
    OutputStream xmlOutputStream = new FileOutputStream(request.getSession().getServletContext()
        .getRealPath("/") + "/xml/signed_hash.xsig");
    IOUtils.copy(httpResponse.getEntity().getContent(), xmlOutputStream);

    xmlOutputStream.close();
}
```

5. Se envía una petición HTTP DELETE a Giltza para eliminar el documento del servidor, utilizando el atributo de sesión *URLignatureProcess*.

```
HttpDelete httpDelete = new HttpDelete(request.getSession().getAttribute("URLignatureProcess")
    .toString());
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

6. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri", "http://localhost:8081/minimalist-oauth-rp/home");
// Atributo para poder descargar el fichero desde el JSP
request.setAttribute("xmlFileName", "signed_hash.xsig");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_xades_signature_end.jsp").forward(request,
response);
```

<jsonString>

Se trata de la variable que contiene el JSON con los parámetros de la firma (incluido el hash) que se va a enviar a Giltza para la realización de la firma.

```
{
  "process_type": "urn:safelayer:eidas:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eidas:policiess:sign:document:xml",
    "parameters": {
      "type": "xades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "detached",
        "nodes_to_sign": [
          {
            "type": "external_reference",
            "uri": "urn:detached",
            "digest_algorithm": "sha256",
            "digest_value": "pay8LuC0dqkEeZ9cqxsZJTviol8AWyBRLwGcKQDyFhk="
          }
        ]
      }
    }
  },
  "ui_locales": ["es_ES"],
  "timestamp": {
    "provider_id": "urn:safelayer:twss:policiess:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}
```

11.2.5. Firma CAdES Attached

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un grant del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parámetros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eidas:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");
```

2. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```

httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/
signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso
de firma
MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
multipartBuilder.setMode(HttpMultipartMode.RFC6532);
// jsonString contiene un JSON con los parametros de la firma
multipartBuilder.addTextBody("process", jsonString,ContentType.APPLICATION_JSON);
// byteArray es el documento que se va a enviar a firmar a Giltza
multipartBuilder.addBinaryBody("document", byteArray,
ContentType.create("application/msword", Consts.UTF_8), binaryFileName);
httpPost.setEntity(multipartBuilder.build());

// Se envía la petición a Giltza
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-
8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject obJson = new JSONObject(newJSONTokener(tokenResponse));

// URLSignatureProcess contiene la URL para eliminar la petición del documento en Giltza
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocument contiene la URL donde se ubica el documento firmado
String URLSignatureProcess = obJson.getString("self");
String urlUserRedirectAuthorization =
obJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");
String URLignedDocument =
obJson.getJSONArray("documents").getJSONObject(0).get("url").toString() +"/content";

```

3. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```

request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLSignatureProcess", URLSignatureProcess);
request.getSession().setAttribute("URLignedDocument", URLignedDocument);
response.sendRedirect(urlUserRedirectAuthorization);

```

4. En el *callback*, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar el documento firmado haciendo uso de la URL indicada en la variable de sesión *URLignedDocument*, mediante una petición HTTP GET.

```

String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = newHttpGet(request.getSession().getAttribute("URLignedDocument")
.toString());

    httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

    CloseableHttpClient httpClient = HttpClientBuilder.create().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse =httpClient.execute(httpGet);

    // Si no hay ningún error, se obtiene una respuesta binaria con el documento firmado. Se
    // copia este documento firmado a un fichero en local
    OutputStream xmlOutputStream = new FileOutputStream(request.getSession().getServletContext()
.getRealPath("/") + "/binary/signed.bin");
    IOUtils.copy(httpResponse.getEntity().getContent(), xmlOutputStream);

    xmlOutputStream.close();
}

```

5. Se envía una petición HTTP DELETE a Giltza para eliminar el documento del servidor, utilizando el atributo de sesión *URLSignatureProcess*.

```
HttpDelete httpDelete = new HttpDelete(request.getSession().getAttribute("URLSignatureProcess")
.toString());
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

6. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri", "http://localhost:8081/minimalist-oauth-rp/home");
// Atributo para poder descargar el fichero desde el JSP
request.setAttribute("binaryFileName", "signed.bin");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_cades_signature_end.jsp").forward(request,
response);
```

<jsonString>

Se trata de la variable que contiene el JSON con los parámetros de la firma que se va a enviar a Giltza para la realización de la firma.

```
{
  "process_type": "urn:safelayer:eid:processes:document:sign",
  "signer": {
    "signature_policy_id": "urn:safelayer:eid:polices:sign:document:cms",
    "parameters": {
      "type": "cades-bes",
      "default_digest_algorithm": "sha256",
      "signature_target": {
        "type": "document",
        "signature_packaging": "enveloping"
      }
    }
  },
  "ui_locales": ["es_ES"],
  "timestamp": {
    "provider_id": "urn:safelayer:tw:polices:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}
```

11.2.6. Firma lotes PAdES con diferente apariencia en la firma visible

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un grant del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eid.izenpe.com/trustedx-authserver/oauth/esignsp/
token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parámetros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eid:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(newJSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String) tokenJson.get("access_token");
```

2. Se crea el recurso para firmar el documento que se proporcionará junto a la definición de firma mediante una petición HTTP POST. En este ejemplo se va a firmar un lote de dos documentos PDF.

```
// jsonObj contiene un JSON con la definicion del proceso defirma
// jsonArrayDocs contiene el parametro 'documents' del JSON
JSONArray jsonArrayDocs = jsonObj.getJSONArray("documents");

InputStream inputStream = null;
for (int i = 0; i < 2; i++) {
    // Diferente apariencia para cada documento PDF
    if (i == 0) {
        // Se recupera el JSON con la apariencia de PDF1
        InputStream =getClass().getClassLoader().getResourceAsStream(applicationProperties
        .getProperty("signature.pdf.batch.appearance1.properties"));

        // El resto de parametros tienen que pasarse en el cuerpo de la peticion para iniciar el
        // proceso de firma
        MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
        multipartBuilder.setMode(HttpMultipartMode.RFC6532);
        multipartBuilder.addTextBody("signers", IOUtils.toString(inputStream),
        ContentType.APPLICATION_JSON);
        // pdfByteArray es el documento PDF que se va a enviar a firmar a Giltza
        multipartBuilder.addBinaryBody("document", pdfByteArray,
        ContentType.create("application/pdf", Consts.UTF_8), "doc_" + i + ".pdf");
        httpPost.setEntity(multipartBuilder.build());
    } else {
        // Se recupera el JSON con la apariencia de PDF2
        InputStream =getClass().getClassLoader().getResourceAsStream(applicationProperties
        .getProperty("signature.pdf.batch.appearance2.properties"));

        // El resto de parametros tienen que pasarse en el cuerpo de la peticion para iniciar el
        // proceso de firma
        MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
        multipartBuilder.setMode(HttpMultipartMode.RFC6532);
        multipartBuilder.addTextBody("signers", IOUtils.toString(inputStream),
        ContentType.APPLICATION_JSON);
        // pdfByteArray es el documento PDF que se va a enviar a firmar a Giltza
        multipartBuilder.addBinaryBody("document", pdfByteArray,
        ContentType.create("application/pdf", Consts.UTF_8), "doc_" + i + ".pdf");
        httpPost.setEntity(multipartBuilder.build());
    }
    httpResponse = httpClient.execute(httpPost);

    reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(),
    "UTF-8"));

    // tokenResponse contiene una estructura JSON
    tokenResponse = reader.readLine();

    // Se parsean los IDs de los documentos al JSON con las propiedades de firma
    try {
        JSONObject obJson = new JSONObject(newJSONTokener(tokenResponse));

        JSONObject jsonObjectId = new JSONObject();
        jsonObjectId.put("id", obJson.getString("id"));
        jsonArrayDocs.put(jsonObjectId);
    } catch (JSONException e) {
        throw new ServletException(e);
    }
}
```

3. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/
signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso
de firma
httpPost.setHeader("Content-Type",ContentType.APPLICATION_JSON.getMimeType());
// jsonObj contiene un JSON con la definición de la firma
HttpEntity entityString = new StringEntity(jsonObj.toString());
httpPost.setEntity(entityString);

// Se envía la petición a Giltza
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-
8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject jsonObj = new JSONObject(newJSONTokener(tokenResponse));

// URLsignatureId contiene el id del proceso de firma
// URLsignatureProcess contiene la URL para eliminar la petición del document en Giltza
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocuments contiene las URL donde se ubican los documentos firmados String
URLsignatureId = jsonObj.getString("id");
String URLsignatureProcess = jsonObj.getString("self");
String urlUserRedirectAuthorization =
jsonObj.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");

ArrayList<String> URLignedDocuments = newArrayList<String>();
for (int i = 0; i < jsonObj.getJSONArray("documents").length(); i++) {
    URLignedDocuments.add(jsonObj.getJSONArray("documents").getJSONObject(i).get("url").toStr
ing() + "/content");
}
```

4. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLsignatureId", URLsignatureId);
request.getSession().setAttribute("URLsignatureProcess", URLsignatureProcess);
request.getSession().setAttribute("URLignedDocuments", URLignedDocuments);
response.sendRedirect(urlUserRedirectAuthorization);
```

5. En el *callback*, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar la información y resultado de los documentos firmados haciendo uso de la URL indicada en la variable de sesión `URLignatureProcess`, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignatureProcess")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse =httpClient.execute(httpGet);

    BufferedReader reader = new BufferedReader(newInputStreamReader(httpResponse.getEntity()
        .getContent(), "UTF-8"));

    // tokenResponse contiene una estructura JSON
    String tokenResponse = reader.readLine();

    // Se obtienen los documentos firmados
    JSONObject obJson = new JSONObject(new JSONTokener(tokenResponse));
    JSONArray jsonObjectDocs =obJson.getJSONArray("documents");
    for (int i = 0; i < jsonObjectDocs.length(); i++){
        httpGet = new HttpGet(jsonObjectDocs.getJSONObject(i).getString("content"));

        httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

        // Se envia la peticion a Giltza
        httpResponse = httpClient.execute(httpGet);

        String fileName = "batch_" + (i + 1) + ".pdf";
        OutputStream pdfOutputStream = newFileOutputStream(request.getSession()
            .getServletContext().getRealPath("/") + "/pdf/signed_" +fileName);

        IOUtils.copy(httpResponse.getEntity().getContent(), pdfOutputStream);
        pdfOutputStream.close();

        // Atributo para poder descargar el fichero desde el JSP
        request.setAttribute("pdfFileName" + (i + 1), fileName);

        [...] // continua en siguiente punto
    }
}
```

6. Continuando en el bucle “for” anterior, se envía una petición HTTP DELETE a Giltza por cada uno de los recursos para eliminarlos del servidor, utilizando el identificador incluido en el JSON.

```
[...] continua del anterior punto

HttpDelete httpDelete = new HttpDelete("https://eidas.izenpe.com/trustedx-resources/esignsp/
v2/documents/" + jsonObjectDocs.getJSONObject(i).getString("id"));
httpDelete.setHeader("Authorization", "Bearer " +applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

7. Se elimina el proceso global de firma de documentos, de nuevo, mediante una petición HTTP DELETE. Para completar la URL de borrado, se hará uso del atributo de sesión `URLignatureId`. Esta llamada se realiza fuera del bucle “for”.

```
HttpDelete httpDelete = new HttpDelete("https://eidas.izenpe.com/trustedx-resources/esignsp/
v2/signer_processes/" + request.getSession().getAttribute("URLignatureId").toString());
httpDelete.setHeader("Authorization", "Bearer " +applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

8. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri","http://localhost:8081/minimalist-oauth-rp/home");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/ document_batch_signature_end.jsp").forward(request,
response);
```

<jsonObj>

Se trata de la variable que contiene el JSON con la definición del proceso de firma que se va a enviar a Giltza.

```
{
  "process_type": "urn:safelayer:eidas:processes:document:sign",
  "documents": [],
  "ui_locales" : ["es_ES"],
  "timestamp" : {
    "provider_id" : "urn:safelayer:twspolicies:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}
```

<inputStream>

Variable que contiene el JSON con la apariencia que va a tener el recurso (documento PDF) que se va a enviar a Giltza para su firma.

Se muestra el JSON que describe los parámetros de firma para el primer documento.

```
[[
  "signature_policy_id": "urn:safelayer:eidas:policiessign:document:pdf",
  "parameters": {
    "type": "pades-bes",
    "default_digest_algorithm": "sha256",
    "location": "Barcelona, Spain",
    "signature_field": {
      "name": "user_signature",
      "location": {
        "page": {
          "number": "last"
        },
        "rectangle": {
          "x": 100,
          "y": 110,
          "height": 150,
          "width": 400
        }
      }
    },
    "appearance": {
      "signature_details": {
        "details": [{
          "type": "subject",
          "title": "Signer Distinguished Name: "
        },
        {
          "type": "location",
          "title": "Signature location: "
        },
        {
          "type": "date",
          "title": "Signature date: "
        }
      ]
    }
  }
]
```

```
    }
  }
}
```

Se muestra el JSON que describe los parámetros de firma para el segundo documento.

```
[{
  "signature_policy_id": "urn:safelayer:eid:sign:document:pdf",
  "parameters": {
    "type": "pades-bes",
    "default_digest_algorithm": "sha256",
    "location": "Vitoria, Spain",
    "signature_field": {
      "name": "user_signature",
      "location": {
        "page": {
          "number": "last"
        },
        "rectangle": {
          "x": 50,
          "y": 110,
          "height": 150,
          "width": 400
        }
      }
    },
    "appearance": {
      "signature_details": {
        "details": [{
          "type": "subject",
          "title": "Subject: "
        },
        {
          "type": "location",
          "title": "Location: "
        },
        {
          "type": "date",
          "title": "Date: "
        }
      ]
    }
  }
}]
```

11.2.7. Firma lotes XAdES Externally Detached a partir de un hash

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un *grant* del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eid.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autentificar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parámetros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eid:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONTokener(tokenResponse));

// Contiene el token con los credenciales para firmar documentos
String applicationAccessToken = (String) tokenJson.get("access_token");
```

2. Se crea el recurso para firmar el documento que se proporcionará junto a la definición de firma mediante una petición HTTP POST. En este ejemplo se va a firmar un lote de dos hashes obtenidos de un documento previo.

```
// jsonObj contiene un JSON con la definición del proceso defirma
// jsonArrayDocs contiene el parametro 'documents' del JSON
JSONArray jsonArrayDocs = jsonObj.getJSONArray("documents");

InputStream inputStream = null;
for (int i = 0; i < 2; i++) {
    // Diferente apariencia para cada documento PDF
    if (i == 0) {
        // Se recupera el JSON con el hash 1
        inputStream=getClass().getClassLoader().getResourceAsStream(applicationProperties
.getProperty("signature.xml.hash1.resource.batch.properties"));
    } else {
        // Se recupera el JSON con el hash 2
        inputStream=getClass().getClassLoader().getResourceAsStream(applicationProperties
.getProperty("signature.xml.hash2.resource.batch.properties"));
    }

    HttpEntity entityString = new StringEntity(IOUtils.toString(inputStream));
    httpPost.setEntity(entityString);

    httpResponse = httpClient.execute(httpPost);

    reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(),
"UTF-8"));

    // tokenResponse contiene una estructura JSON
    tokenResponse = reader.readLine();

    // Se parsean los IDs de los documentos al JSON con las propiedades de firma
    try {
        JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

        JSONObject jsonObjectId = new JSONObject();
        jsonObjectId.put("id", objJson.getString("id"));
        jsonArrayDocs.put(jsonObjectId);
    } catch (JSONException e) {
        throw new ServletException(e);
    }
}
```

3. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/
signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el proceso
de firma
httpPost.setHeader("Content-Type",ContentType.APPLICATION_JSON.getMimeType());
// jsonObj contiene un JSON con la definición de la firma
HttpEntity entityString = new StringEntity(jsonObj.toString());
httpPost.setEntity(entityString);

// Se envia la petición a Giltza
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-
8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

// URLsignatureId contiene el id del proceso de firma
// URLsignatureProcess contiene la URL para eliminar la petición del document en Giltza
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocuments contiene las URL donde se ubican los documentos firmados String
URLsignatureId = objJson.getString("id");
String URLsignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");

ArrayList<String> URLignedDocuments = newArrayList<String>();
for (int i = 0; i < objJson.getJSONArray("documents").length(); i++) {
    URLignedDocuments.add(objJson.getJSONArray("documents").getJSONObject(i).get("url").toStr
ing() + "/content");
}
```

4. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLsignatureId", URLsignatureId);
request.getSession().setAttribute("URLsignatureProcess", URLsignatureProcess);
request.getSession().setAttribute("URLignedDocuments", URLignedDocuments);
response.sendRedirect(urlUserRedirectAuthorization);
```

5. En el *callback*, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar la información y resultado de los documentos firmados haciendo uso de la URL indicada en la variable de sesión *URLignatureProcess*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLignatureProcess")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse =httpClient.execute(httpGet);

    BufferedReader reader = new BufferedReader(newInputStreamReader(httpResponse.getEntity()
        .getContent(), "UTF-8"));

    // tokenResponse contiene una estructura JSON
    String tokenResponse = reader.readLine();

    // Se obtienen los documentos firmados
    JSONObject obJson = new JSONObject(new JSONTokener(tokenResponse));
    JSONArray jsonObjectDocs =obJson.getJSONArray("documents");
    for (int i = 0; i < jsonObjectDocs.length(); i++){
        httpGet = new HttpGet(jsonObjectDocs.getJSONObject(i).getString("content"));

        httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

        // Se envia la peticion a Giltza
        httpResponse = httpClient.execute(httpGet);

        String fileName = "batch_" + (i + 1) + ".xsig";
        OutputStream xmlOutputStream = newFileOutputStream(request.getSession()
            .getServletContext().getRealPath("/") + "/xml/signed_" +fileName);

        IOUtils.copy(httpResponse.getEntity().getContent(), xmlOutputStream);
        xmlOutputStream.close();

        // Atributo para poder descargar el fichero desde el JSP
        request.setAttribute("xmlFileName" + (i + 1), fileName);

        [...] // continua en siguiente punto
    }
}
```

6. Continuando en el bucle *“for”* anterior, se envía una petición HTTP DELETE a Giltza por cada uno de los recursos para eliminarlos del servidor, utilizando el identificador incluido en el JSON.

```
[...] continua del anterior punto

HttpDelete httpDelete = new HttpDelete("https://eidas.izenpe.com/trustedx-resources/esignsp/
v2/documents/" + jsonObjectDocs.getJSONObject(i).getString("id"));
httpDelete.setHeader("Authorization", "Bearer " +applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

7. Se elimina el proceso global de firma de hashes, de nuevo, mediante una petición HTTP DELETE. Para completar la URL de borrado, se hará uso del atributo de sesión *URLignatureId*. Esta llamada se realiza fuera del bucle *“for”*.

```
HttpDelete httpDelete = new HttpDelete("https://eidas.izenpe.com/trustedx-resources/esignsp/
v2/signer_processes/" + request.getSession().getAttribute("URLignatureId").toString());
httpDelete.setHeader("Authorization", "Bearer " +applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

8. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri","http://localhost:8081/minimalist-oauth-rp/home");  
  
// Se muestra la pagina JSP de exito  
request.getRequestDispatcher("/WEB-INF/views/ document_batch_signature_end.jsp").forward(request,  
response);
```

<jsonObj>

Se trata de la variable que contiene el JSON con la definición del proceso de firma que se va a enviar a Giltza.

```
{  
  "process_type": "urn:safelayer:eid:processes:document:sign",  
  "documents": [],  
  "ui_locales" : ["es_ES"],  
  "timestamp" : {  
    "provider_id" : "urn:safelayer:tw:polices:generation:esigsp"  
  },  
  "finish_callback_url": "http://localhost"  
}
```

<inputStream>

Variable que contiene el JSON con la apariencia que va a tener el recurso (hash) que se va a enviar a Giltza para su firma.

Se muestra el JSON que describe los parámetros de firma para el primer hash.

```
[[  
  "signature_policy_id": "urn:safelayer:eid:polices:sign:document:xml",  
  "parameters": {  
    "type": "xades-bes",  
    "signature_target": {  
      "type": "document",  
      "signature_packaging": "detached",  
      "nodes_to_sign": [  
        {  
          "type": "external_reference",  
          "uri": "urn:detached",  
          "digest_algorithm": "sha256",  
          "digest_value": "W2aub2w43xaF8mjVr3N+qUWzjS0rip8xpeu0jDeJfgw="  
        }  
      ]  
    }  
  }  
]]
```

Se muestra el JSON que describe los parámetros de firma para el segundo hash.

```
[[  
  "signature_policy_id": "urn:safelayer:eid:polices:sign:document:xml",  
  "parameters": {  
    "type": "xades-bes",  
    "signature_target": {  
      "type": "document",  
      "signature_packaging": "detached",  
      "nodes_to_sign": [  
        {  
          "type": "external_reference",  
          "uri": "urn:detached",  
          "digest_algorithm": "sha256",  
          "digest_value": "pay8LuC0dqkEeZ9cqxsZJTviol8AWyBRLwGcKQDyFhk="  
        }  
      ]  
    }  
  }  
]]
```

11.2.8. Firma lotes combinando diferentes perfiles de firma (PAdES + XAdES)

1. Se obtiene el token OAuth 2.0 mediante una petición HTTP POST, haciendo uso de un grant del tipo *client_credentials*.

```
HttpPost httpPost = new HttpPost("https://eid.izenpe.com/trustedx-authserver/oauth/esignsp/token");

// Para autenticar la petición, utilizar el API_KEY como cabecera HTTP
String apiKey = applicationProperties.getProperty("apiKey");
httpPost.setHeader("Authorization", "Basic " + apiKey);

// El resto de parámetros se envían en el cuerpo de la petición
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("grant_type", "client_credentials"));
params.add(new BasicNameValuePair("scope", "urn:safelayer:eid:sign:process:document"));
httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
CloseableHttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity entity = httpResponse.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(entity.getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
String tokenResponse = reader.readLine();
JSONObject tokenJson = new JSONObject(new JSONObject(tokenResponse));

// Contiene el token con las credenciales para firmar documentos
String applicationAccessToken = (String)tokenJson.get("access_token");
```

2. Se crea el recurso para firmar el documento que se proporcionará junto a la definición de firma mediante una petición HTTP POST. En este ejemplo se va a firmar un lote combinando un documento PDF y un documento XML.

```
// jsonObj contiene un JSON con la definicion del proceso defirma
// jsonArrayDocs contiene el parametro 'documents' del JSON
JSONArray jsonArrayDocs = jsonObj.getJSONArray("documents");

InputStream inputStream = null;
for (int i = 0; i < 2; i++) {
    if (i == 0) {
        // Se recupera el JSON correspondiente al PDF
        inputStream = getClass().getClassLoader().getResourceAsStream(applicationProperties
        .getProperty("signature.pdfXml.batch.sign1.properties"));

        // El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el
        // proceso de firma
        MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
        multipartBuilder.setMode(HttpMultipartMode.RFC6532);
        multipartBuilder.addTextBody("signers", IOUtils.toString(inputStream),
        ContentType.APPLICATION_JSON);
        // pdfByteArray es el documento PDF que se va a enviar a firmar a Giltza
        multipartBuilder.addBinaryBody("document", pdfByteArray,
        ContentType.create("application/pdf", Consts.UTF_8), "doc_" + i + ".pdf");
        httpPost.setEntity(multipartBuilder.build());
    } else {
        // Se recupera el JSON correspondiente al XML
        inputStream = getClass().getClassLoader().getResourceAsStream(applicationProperties
        .getProperty("signature.pdfXml.batch.sign2.properties"));

        // El resto de parametros tienen que pasarse en el cuerpo de la petición para iniciar el
        // proceso de firma
        MultipartEntityBuilder multipartBuilder = MultipartEntityBuilder.create();
        multipartBuilder.setMode(HttpMultipartMode.RFC6532);
        multipartBuilder.addTextBody("signers", IOUtils.toString(inputStream),
        ContentType.APPLICATION_JSON);
        // xmlByteArray es el documento XML que se va a enviar a firmar a Giltza
        multipartBuilder.addBinaryBody("document", xmlByteArray, ContentType.create("text/xml",
        Consts.UTF_8), "doc_" + i + ".xml");
        httpPost.setEntity(multipartBuilder.build());
    }

    httpResponse = httpClient.execute(httpPost);

    reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(),
    "UTF-8"));

    // tokenResponse contiene una estructura JSON
    tokenResponse = reader.readLine();

    // Se parsean los IDs de los documentos al JSON con las propiedades de firma
    try {
        JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

        JSONObject jsonObjectId = new JSONObject();
        jsonObjectId.put("id", objJson.getString("id"));
        jsonArrayDocs.put(jsonObjectId);
    } catch (JSONException e) {
        throw new ServletException(e);
    }
}
}
```

3. Se inicializa el proceso de firma mediante una petición HTTP POST. Para ello, se va a utilizar el token de acceso obtenido en el primer paso.

```
httpPost = new HttpPost("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes");

// Para autenticar la petición con la aplicación, utilizar el nuevo token obtenido
httpPost.setHeader("Authorization", "Bearer " +applicationAccessToken);

// El resto de parámetros tienen que pasarse en el cuerpo de la petición para iniciar el proceso de firma
httpPost.setHeader("Content-Type",ContentType.APPLICATION_JSON.getMimeType());
// jsonObj contiene un JSON con la definición de la firma
HttpEntity entityString = new StringEntity(jsonObj.toString());
httpPost.setEntity(entityString);

// Se envía la petición a Giltza
httpResponse =httpClient.execute(httpPost);

reader = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent(), "UTF-8"));

// tokenResponse contiene una estructura JSON
tokenResponse = reader.readLine();

JSONObject objJson = new JSONObject(newJSONTokener(tokenResponse));

// URLSignatureId contiene el id del proceso de firma
// URLSignatureProcess contiene la URL para eliminar la petición del documento en Giltza
// urlUserRedirectAuthorization contiene el id del proceso de firma
// URLignedDocuments contiene las URL donde se ubican los documentos firmados String
URLSignatureId = objJson.getString("id");
String URLSignatureProcess = objJson.getString("self");
String urlUserRedirectAuthorization =
objJson.getJSONObject("tasks").getJSONArray("pending").getJSONObject(0).getString("url");

ArrayList<String> URLignedDocuments = newArrayList<String>();
for (int i = 0; i < objJson.getJSONArray("documents").length(); i++) {
    URLignedDocuments.add(objJson.getJSONArray("documents").getJSONObject(i).get("url").toString() + "/content");
}
```

4. Se guardan en sesión las variables recién obtenidas de la respuesta JSON devuelta por Giltza y se redirecciona el navegador para ejecutar el proceso de firma.

```
request.getSession().setAttribute("applicationAccessToken", applicationAccessToken);
request.getSession().setAttribute("URLSignatureId", URLSignatureId);
request.getSession().setAttribute("URLSignatureProcess", URLSignatureProcess);
request.getSession().setAttribute("URLignedDocuments", URLignedDocuments);
response.sendRedirect(urlUserRedirectAuthorization);
```

5. En el *callback*, y sólo si el proceso de firma se ha ejecutado con éxito, se procede a recuperar la información y resultado de los documentos firmados haciendo uso de la URL indicada en la variable de sesión *URLSignatureProcess*, mediante una petición HTTP GET.

```
String status = request.getParameter("status").toString();
if ("finished".equals(status)) {
    HttpGet httpGet = new HttpGet(request.getSession().getAttribute("URLSignatureProcess")
        .toString());

    httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

    CloseableHttpClient httpClient = HttpClients.custom().setSslContext(sslContext).build();
    CloseableHttpResponse httpResponse =httpClient.execute(httpGet);

    BufferedReader reader = new BufferedReader(newInputStreamReader(httpResponse.getEntity()
        .getContent(), "UTF-8"));

    // tokenResponse contiene una estructura JSON
    String tokenResponse = reader.readLine();

    // Se obtienen los documentos firmados
    JSONObject obJson = new JSONObject(new JSONTokener(tokenResponse));
    JSONArray jsonObjectDocs =obJson.getJSONArray("documents");
    for (int i = 0; i < jsonObjectDocs.length(); i++){
        httpGet = new HttpGet(jsonObjectDocs.getJSONObject(i).getString("content"));

        httpGet.setHeader("Authorization", "Bearer " +applicationAccessToken);

        // Se envia la petición a Giltza
        httpResponse = httpClient.execute(httpGet);

        // Se copia este documento firmado a un fichero en local
        String fileName = null;
        if (i == 0) {
            // Si no hay ningun error, se obtiene una respuesta"application/pdf"
            fileName = "batch_" + (i + 1) + ".pdf";
            pdfOutputStream = newFileOutputStream(request.getSession().getServletContext()
                .getRealPath("/") + "/pdf/signed_" + fileName);
        } else {
            // Si no hay ningun error, se obtiene una respuesta"text/xml"
            fileName = "batch_" + (i + 1) + ".xsig";
            pdfOutputStream = newFileOutputStream(request.getSession().getServletContext()
                .getRealPath("/") + "/xml/signed_" + fileName);
        }

        IOUtils.copy(httpResponse.getEntity().getContent(), pdfOutputStream);
        pdfOutputStream.close();

        // Atributo para poder descargar el fichero desde el JSP
        request.setAttribute("pdfFileName" + (i + 1), fileName);

        [...] // continua en siguiente punto
    }
}
```

6. Continuando en el bucle *“for”* anterior, se envía una petición HTTP DELETE a Giltza por cada uno de los recursos para eliminarlos del servidor, utilizando el identificador incluido en el JSON.

```
[...] continua del anterior punto

HttpDelete httpDelete = new HttpDelete("https://eid.izenpe.com/trustedx-resources/esignsp/
v2/documents/" + jsonObjectDocs.getJSONObject(i).getString("id"));
httpDelete.setHeader("Authorization", "Bearer " +applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

7. Se elimina el proceso global de firma de documentos, de nuevo, mediante una petición HTTP DELETE. Para completar la URL de borrado, se hará uso del atributo de sesión *URLSignatureId*. Esta llamada se realiza fuera del bucle "for".

```
HttpDelete httpDelete = new HttpDelete("https://eidas.izenpe.com/trustedx-resources/esignsp/v2/signer_processes/" + request.getSession().getAttribute("URLSignatureId").toString());
httpDelete.setHeader("Authorization", "Bearer " + applicationAccessToken);
httpResponse = httpClient.execute(httpDelete);
```

8. Por último, se muestra la página JSP de éxito, pasando los atributos necesarios.

```
request.setAttribute("appHomeRedirectUri", "http://localhost:8081/minimalist-oauth-rp/home");

// Se muestra la pagina JSP de exito
request.getRequestDispatcher("/WEB-INF/views/document_batch_signature_end.jsp").forward(request, response);
```

<jsonObj>

Se trata de la variable que contiene el JSON con la definición del proceso de firma que se va a enviar a Giltza.

```
{
  "process_type": "urn:safelayer:eidas:processes:document:sign",
  "documents": [],
  "ui_locales" : ["es_ES"],
  "timestamp" : {
    "provider_id" : "urn:safelayer:twspolicies:generation:esignsp"
  },
  "finish_callback_url": "http://localhost"
}
```

<inputStream>

Variable que contiene el JSON con la apariencia que va a tener el recurso (documento PDF o documento XML) que se va a enviar a Giltza para su firma.

Se muestra el JSON que describe los parámetros de firma para el documento PDF.

```
{
  "signature_policy_id": "urn:safelayer:eidas:policies:sign:document:pdf",
  "parameters": {
    "type": "pades-bes",
    "default_digest_algorithm": "sha256",
    "location": "Barcelona, Spain",
    "signature_field": {
      "name": "user_signature",
      "location": {
        "page": {
          "number": "last"
        },
        "rectangle": {
          "x": 100,
          "y": 110,
          "height": 150,
          "width": 400
        }
      }
    },
    "appearance": {
      "signature_details": {
        "details": [{
          "type": "subject",
          "title": "Signer Distinguished Name: "
        }],
      }
    }
  }
}
```

```

        "type": "location",
        "title": "Signature location: "
    },
    {
        "type": "date",
        "title": "Signature date: "
    }
]
}
}
]]

```

Se muestra el JSON que describe los parámetros de firma para el documento XML.

```

[[
  "signature_policy_id": "urn:safelayer:eid:policiés:sign:document:xml",
  "parameters": {
    "type": "xades-bes",
    "default_digest_algorithm": "sha256",
    "signature_target": {
      "type": "document",
      "signature_packaging": "enveloped",
      "nodes_to_sign": [
        {
          "type": "document_reference",
          "xpath": "/"
        }
      ],
      "signature_placement": {
        "type": "last_child_of",
        "xpath": "/*"
      }
    }
  }
}
]]

```

12. Ejemplos de verificación de firmas mediante Zain

Debido a la posible complejidad que puede llevar la verificación de algunos tipos de firmas generadas en Giltza contra la plataforma de firma Zain, en este apartado se van a describir las peticiones SOAP que hay que crear para que el resultado sea el esperado.

12.1. Firma XAdES Enveloping con elemento Manifest

En este tipo de firmas, para indicar que sólo se verifique la firma aportada y que no se verifiquen las referencias del propio Manifest, habrá que indicar en la petición SOAP el siguiente elemento:

```
smartVerifyRequest.setXmlIncludeManifests(true);
```

En caso contrario, si se desea verificar la firma aportada y las referencias del propio Manifest, habrá que indicar en la petición SOAP el siguiente elemento:

```
smartVerifyRequest.setXmlIncludeManifests(false);
```

Petición SOAP de verificación

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <VerifyRequest
      xmlns="http://www.docs.oasis-open.org/dss/2004/06/oasis-dss-1.0-
      core-schema-wd-27.xsd"
      Profile="urn:safelayer:tws:dss:1.0:profiles:xades:1.0:verify"
      RequestID="8c4af138578accb9b2d3">
      <OptionalInputs>
        <Language xsi:type="xsd:language">es</Language>
        <ns1:AddCertificateValues
          xmlns:ns1="http://www.safelayer.com/TWS"

```

```

binary="false"
                                xsi:type="ns1:AddCertificateValuesType" />
                                <ns2:AddRevocationValues
binary="false"
                                xmlns:ns2="http://www.safelayer.com/TWS"
                                xsi:type="ns2:AddRevocationValuesType" />
                                <ns3:AddTimeStampValues
binary="false" />
                                xmlns:ns3="http://www.safelayer.com/TWS"
                                <ns4:AddSignatureForm
xsi:type="xsd:string" />
                                xmlns:ns4="http://www.safelayer.com/TWS"
                                (<ns5:IncludeManifests
                                xmlns:ns5="http://www.safelayer.com/TWS" />)
                                </OptionalInputs>
                                <SignatureObject>
                                <ns5:Base64XMLSignature
                                xmlns:ns5="http://www.safelayer.com/TWS"
                                xsi:type="ns5:base64Binary">PD9...ZT4=
                                </ns5:Base64XMLSignature>
                                </SignatureObject>
                                <InputDocuments>
                                <DocumentHash>
                                <ns6:DigestMethod
                                xmlns:ns6="http://www.w3.org/2000/09/xmldsig#"
                                Algorithm="http://www.w3.org/2001/04/xmenc#sha512"
                                xsi:type="ns6:DigestMethodType" />
                                <ns7:DigestValue
                                xmlns:ns7="http://www.w3.org/2000/09/xmldsig#"
                                xsi:type="ns7:DigestValueType">9wo...2hM0=</ns7:DigestValue>
                                </DocumentHash>
                                </InputDocuments>
                                </VerifyRequest>
                                </soapenv:Body>
</soapenv:Envelope>

```

Para crear el nodo *DocumentHash* en la petición SOAP de verificación, la API de Zain dispone de dos opciones igualmente válidas. La primera se muestra a continuación:

```

smartVerifyRequest.setInputRefUri("id-manifest:1");
smartVerifyRequest.setInputHashAlgorithm(Constants.DigestAlgorithm.SHA512);
smartVerifyRequest.setInputHashDigest("9wo...2hM0=");

```

La segunda opción es más eficiente en cuanto a rapidez de implementación se refiere:

```

smartVerifyRequest.setInputHashDigestXadesItem(0, "9wo...2hM0=",
Constants.DigestAlgorithm.SHA512, "id-manifest:1");

```

12.2. Firma XAdES Enveloping con varios elementos Manifest

En este tipo de firmas, para indicar que sólo se verifique la firma aportada y que no se verifiquen las referencias de los propios Manifest, habrá que indicar en la petición SOAP el siguiente elemento:

```

smartVerifyRequest.setXmlIncludeManifests(true);

```

En caso contrario, si se desea verificar la firma aportada y las referencias de los propios Manifest, habrá que indicar en la petición SOAP el siguiente elemento:

```

smartVerifyRequest.setXmlIncludeManifests(false);

```

Petición SOAP de verificación

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <VerifyRequest
      xmlns="http://www.docs.oasis-open.org/dss/2004/06/oasis-dss-1.0-
core-schema-wd-27.xsd"
      Profile="urn:safelayer:tws:dss:1.0:profiles:xades:1.0:verify"
      RequestID="970cf0e73306c14ecba2">
      <OptionalInputs>
        <Language xsi:type="xsd:language">es</Language>
        <ns1:AddCertificateValues
          xmlns:ns1="http://www.safelayer.com/TWS"
          xsi:type="ns1:AddCertificateValuesType" />
        <ns2:AddRevocationValues
          xmlns:ns2="http://www.safelayer.com/TWS"
          xsi:type="ns2:AddRevocationValuesType" />
        <ns3:AddTimeStampValues
          xmlns:ns3="http://www.safelayer.com/TWS"
          xsi:type="ns3:AddTimeStampValuesType" />
        <ns4:AddSignatureForm
          xmlns:ns4="http://www.safelayer.com/TWS"
          xsi:type="xsd:string" />
        (<ns5:IncludeManifests
          xmlns:ns5="http://www.safelayer.com/TWS" />)
      </OptionalInputs>
      <SignatureObject>
        <ns5:Base64XMLSignature
          xmlns:ns5="http://www.safelayer.com/TWS"
          xsi:type="ns5:base64Binary">PGR...ZT4=
        </ns5:Base64XMLSignature>
      </SignatureObject>
      <InputDocuments>
        <ns6:DocumentItem
          xmlns:ns6="http://www.safelayer.com/TWS"
          RefURI="id-manifest:1">
            <DocumentHash>
              <ns7:DigestMethod
                xmlns:ns7="http://www.w3.org/2000/09/xmldsig#"
                Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"
                xsi:type="ns7:DigestMethodType" />
              <ns8:DigestValue
                xmlns:ns8="http://www.w3.org/2000/09/xmldsig#"
                xsi:type="ns8:DigestValueType">uGX.../c=
              </ns8:DigestValue>
            </DocumentHash>
          </ns6:DocumentItem>
          <ns9:DocumentItem
            xmlns:ns9="http://www.safelayer.com/TWS"
            RefURI="id-manifest:2">
              <DocumentHash>
                <ns10:DigestMethod
                  xmlns:ns10="http://www.w3.org/2000/09/xmldsig#"
                  Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"
                  xsi:type="ns10:DigestMethodType" />
                <ns11:DigestValue
                  xmlns:ns11="http://www.w3.org/2000/09/xmldsig#"
                  xsi:type="ns11:DigestValueType">W2a...fgw=
                </ns11:DigestValue>
              </DocumentHash>
            </ns9:DocumentItem>
          </InputDocuments>
        </VerifyRequest>
      </soapenv:Body>
    </soapenv:Envelope>

```

Para crear el nodo *DocumentHash* en la petición SOAP de verificación, la API de Zain dispone de la siguiente opción:

```

smartVerifyRequest.setInputHashDigestXadesItem(0, "uGX.../c=",
Constants.DigestAlgorithm.SHA512, "id-manifest:1");

```

```

smartVerifyRequest.setInputHashDigestXadesItem(1, "W2a...fgw=",
Constants.DigestAlgorithm.SHA512, "id-manifest:2");

```

13. Referencias

Las referencias utilizadas en el presente documento son las siguientes:

- Tags for Identifying Languages (RFC 5646)
<https://tools.ietf.org/html/rfc5646>
- Update to MIME regarding "charset" Parameter Handling in Textual Media Types (RFC 6657)
<https://tools.ietf.org/html/rfc6657>
- The OAuth 2.0 Authorization Framework (RFC 6749)
<https://tools.ietf.org/html/rfc6749>
- The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC 6750)
<https://tools.ietf.org/html/rfc6750>
- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (RFC 7231)
<https://tools.ietf.org/html/rfc7231>
- OpenID Connect1.0
<http://openid.net/connect/>
- Assertions and Protocols for the OASIS Security Assertion MarkupLanguage (SAML) V2.0. OASIS
<https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS
<http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf>
- Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 . OASIS
<http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf>
- Language Codes (ISO639)
<https://www.iso.org/iso-639-language-codes.html>

- XML Advanced Electronic Signatures, XAdES (ETSI TS 101 903v1.3.2)
http://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.03.02_60/ts_101903v010302p.pdf
- Algoritmos de canonicalización (Algoritmo inclusivo)
<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- Algoritmos de canonicalización (Algoritmo inclusivo con comentarios)
<http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments>
- Algoritmos de canonicalización (Algoritmo exclusivo)
<http://www.w3.org/2001/10/xml-exc-c14n#>
- Algoritmos de canonicalización (Algoritmo exclusivo con comentarios)
<http://www.w3.org/2001/10/xml-exc-c14n#WithComments>
- Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 2: PAdES Basic - Profile based on ISO 32000-1 (ETSI TS 102 778-2v1.2.1)
http://www.etsi.org/deliver/etsi_ts/102700_102799/10277802/01.02.01_60/ts_10277802v010201p.pdf
- Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced - PAdES-BES and PAdES-EPES Profiles (ETSI TS 102 778-3 v1.2.1)
http://www.etsi.org/deliver/etsi_ts/102700_102799/10277803/01.02.01_60/ts_10277803v010201p.pdf
- Cryptographic Message Syntax; CMS (RFC3852)
<https://tools.ietf.org/html/rfc3852>
- Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAAdES); CAAdES-BES and CAAdES-EPES Profiles (ETSI TS 101 733 v2.2.1)
https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf
- Estándar PKCS#11 (Interfaz de dispositivo criptográfico)
<https://web.archive.org/web/20061210143442/http://www.rsasecurity.com/rsalabs/node.asp?id=2133>

- Estándar PKCS#12 (Sintaxis de intercambio de información personal)
<https://web.archive.org/web/20061210143421/http://www.rsasecurity.com/rsalabs/node.asp?id=2138>
- The OAuth 2.0 Authorization Framework (RFC6749)
<https://www.ietf.org/rfc/rfc6749.txt>