

Plataforma Cl@ve 2.0

Manual de Integración (Java, PHP y ASP.NET) para Proveedores de Servicios

Revisión 1.3
08/06/2018

ÍNDICE:

1. Introducción 5

2. Alcance 5

3. Modelo funcional 6

3.1. Flujo de negocio SAML2 7

3.2. Peticiones SAML2 8

3.3. Respuestas SAML2 8

3.4. Calidad de la credencial (LoA) y atributos personales 9

4. Elementos que componen la arquitectura 10

4.1. Proveedor de Servicios (Service Provider o SP) 10

4.2. Proveedor de identidad (Identity Provider o IdP) 10

4.3. Pasarela Cl@ve (proxy entre SPs e IdPs) 10

5. Proceso de alta y requisitos de acceso 11

6. Paquetes de integración para los SPs 12

6.1. Deshabilitar Proveedores de Identificación en la pasarela Cl@ve 12

6.2. Atributo RelayState en la petición SAML 13

6.3. Identificación mediante certificado 13

6.4. Paquete de integración Java 17

6.4.1. Estrategias de integración 18

6.4.2. Ficheros de propiedades 19

6.4.3. Descripción de los ficheros de configuración de un SP 19

6.4.4. Preparación del entorno de desarrollo 20

6.4.5. Certificados 21

6.4.6. Dependencias a importar 21

6.4.7. Exposición de servicios 23

6.4.8. Generación de un ticket SAML2 24

6.4.9. Validación de una respuesta SAML2 25

6.4.9.1. Atributos recibidos en la respuesta 26

6.4.10. Generación de un ticket SAML2 de petición de Logout 27

6.4.11. Validación de una respuesta SAML2 de logout 27

6.4.12. Guía de integración de SP 28

6.4.13. Información adicional 28

6.5. Paquete de integración C# y ASP.NET 31

6.5.1. Ficheros de propiedades 32

6.5.2. Certificados ... 32
6.5.3. Dependencias a importar ... 33
6.5.4. Generación de un ticket SAML ... 33
6.5.5. Validación de una respuesta ... 38
6.5.6. Descifrado de una respuesta ... 41
6.5.7. Log out ... 42
6.6. Paquete de integración PHP ... 45
6.6.1. Ficheros de configuración ... 45
6.6.2. Certificados ... 46
6.6.3. Librerías utilizadas ... 47
6.6.4. Recogida de datos previa de las fuentes de autenticación ... 47
6.6.5. Construcción de la petición de autenticación ... 48
6.6.6. Construcción de la petición de Logout ... 50
6.6.7. Construcción del ticket SAML ... 50
6.6.8. Validación de una respuesta ... 54
6.6.9. Validación de una respuesta de logout ... 60
7. Pasos durante la ejecución de una prueba ... 61
7.1. Validar Request ... 63
7.2. Validar respuesta ... 64
8. Información de conexión a la pasarela Cl@ve ... 65
9. Anexo ... 66
9.1. Algoritmos criptográficos empleados ... 67
9.1.1. Cálculo de *digest* ... 67
9.1.2. Cifrado simétrico ... 68
9.1.3. Cifrado simétrico de tipo AES (Rijndael) ... 68
9.1.4. Modo de cifrado GCM (Galois Counter Mode) ... 70
9.1.5. Cifrado asimétrico ... 71
9.1.6. Cifrado RSA ... 72
9.1.7. SSL/TLS ... 73
9.1.8. Firma y cifrado en el token SAML2 ... 74
9.2. Filtros WEB y configuración de seguridad ... 76
9.2.1. Gestión de ataques de denegación de servicio ... 78
9.2.2. Ataques CSRF ... 79
9.3. Ataques en el XML ... 79

10. Glosario de términos 80
11. Enlaces de interés 80

1. Introducción

El presente documento define e ilustra el proceso de **integración con la plataforma de autenticación federada Cl@ve** por parte de proveedores de servicios de administración electrónica**.**

La implementación de la plataforma Cl@ve se ha basado en las especificaciones técnicas eIDAS, estándar europeo que organiza y asegura la delegación de las funciones de autenticación de un proveedor de servicios, haciendo uso de auditados procesos criptográficos que incluyen firma electrónica y cifrado.

Estas especificaciones técnicas, que son un perfil basado en el estándar SAML 2.0 y utilizan el concepto de identidad federada, permiten que las aplicaciones no necesiten almacenar los datos de los usuarios y soliciten los datos de autenticación y datos referentes a las identidades a una entidad que alberga y custodia los mismos, proporcionando un sistema de SSO.

Para la comunicación entre proveedores de servicios y proveedores de identidades, se incluye un intermediador central que simplifica el proceso de confianza electrónica y audita el proceso.

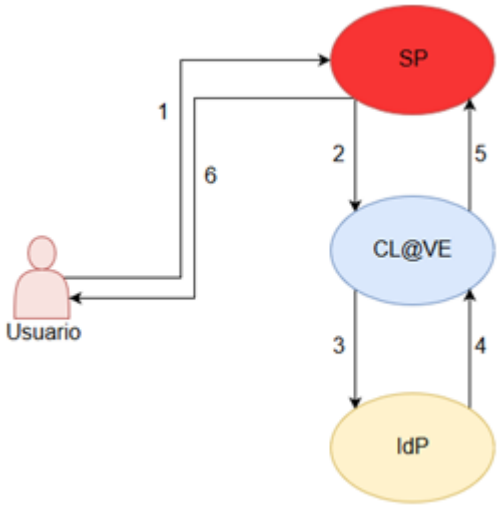
2. Alcance

El presente documento pretende en un primer lugar definir de una forma funcional, de manera que se comprenda de forma concisa, el nuevo modelo de Cl@ve, indicando cada uno de los componetes que forman esta arquitectura. Para posteriormente, describir de forma minuciosa las llamadas y flujos que conlleva la integración.

Se hace énfasis en los paquetes de integración basados en Java para poder integrar por parte del personal técnico.

3. Modelo funcional

Cuando un ciudadano desea acceder a un determinado servicio (por ejemplo, a la oficina electrónica de una administración que hace uso de los sevicios que proporciona Clave) lo primero que hace es conectarse a la aplicación del proveedor de servicios (SP), el cual requiere que se autentique *(paso 1, ver figura a continuación)*.



La necesidad de autenticación del SP se delega en un sistema externo (nodo Cl@ve), por lo que es necesario enviar al usuario hacia dicho sistema externo, lo cual implica cumplir con dos requisitos básicos.

- Que el SP esté dado de alta en el sistema externo. Esto se consigue mediante un trámite de alta previo que realiza el SP con el órgano gestor del nodo Clave. La validación que subyace para asegurar que el SP está dado de alta consiste en una validación mutua a través de tickets SAML que se intercambian entre cada una de las partes. Estos tickets SAML contienen un XML firmado donde se especifican las credenciales públicas del SP en cuestión. Estas credenciales deben estar acreditadas por un tercero de confianza.
- Que el usuario porte un ticket SAML2 firmado por el SP donde se solicita una autenticación en el nodo Clave.

Se firma un ticket SAML2 para que el usuario, mediante una redirección, de tipo POST o de tipo Redirect, vaya al Intermediador. El usuario porta en una cabecera HTML el ticket SAML2 para que al entrar en el servidor, éste lo reciba como parámetro “SAMLRequest” *(paso 2)*.

Seguidamente, el intermediador realiza las tareas de validación del ticket SAML2 y muestra al usuario una página web donde se pide que se escoja el IdP en el que se quiere realizar la autenticación.

Después, en función del IdP seleccionado por el propio usuario, se va al IdP en cuestión *(paso 3)* donde presentará su credencial personal para ser identificado.

Ya con los datos del usuario, se sigue el camino contrario: el IdP responde al intermediador *(paso 4)*, que a su vez responde al proveedor de sericios *(paso 5)*, con un ticket que contiene una firma electrónica y un contenido cifrado. Dentro de este contenido cifrado se encuentran los datos del ciudadano que se han recuperado en el IdP.

Esta funcionalidad aporta un nivel de seguridad y confianza adicional en el tránsito de la información del usuario, ya que los SPs no se comunican directamente con los IdPs, sino que siempre existen unos elementos intermedios que protegen y aseguran los datos solicitados y entregados. El proveedor de servicio utiliza este ticket como fuente de información veraz y procede a iniciar la sesión del usuario en sus sistemas (Paso 6).

En resumen, se parte de una solicitud de autenticación firmada por el SP, la plataforma Cl@ve procesa la credencial presentada y prepara una respuesta firmada que contiene los datos del usuario, sin necesidad de que el proveedor de servicios tenga que implementar lógica para la introducción de credenciales de usuario ni necesite conexión con fuentes de datos de identidad.

3.1. Flujo de negocio SAML2

El protocolo SAML2 (Security Assertion Markup Language 2.0) es un estándar propuesto por OASIS para el intercambio de datos de autenticación y autorización (basado en un estándar de XML) entre entidades conocidas como proveedores de identidad (IdP) y proveedores de servicios (SP).

Dada las características de seguridad que requiere la protección de la identidad digital se realizan dos procesos sobre los mensajes SAML2:

- Se firman digitalmente los mensajes en el proceso de intercambio de información.
- Los datos sensibles del usuario quedan cifrados en los mensajes SAML2 empleado la clave pública presenteen la firma electrónica del ticket SAML de origen, para que sean legible sólo por el poseedor de la clave privada, es decir, el destino al que van dirigidas.

La firma electrónica y el cifrado implican:

- La integridad del mensaje: No hay alteraciones en su contenido.
- La autenticidad del mensaje: Una autoridad asegura mediante evidencias que el mensaje es auténtico. Para ello se emplea un certificado electrónico cualificado.
- No repudio: El mensaje tiene validez como prueba de que la comunicación se produjo.
- Cifrado: Se asegura la información sensible del contenido de forma que solo el destinatario y ningún otro pueda leerlo.
- El protocolo además permite que diferentes sistemas puedan hablar el mismo idioma. Es decir, permite la interoperabilidad entre sistemas heterogéneos.

En un ciclo de autenticación intervienen dos elementos de SAML2: las aserciones de petición (AuthnRequest) y respuesta (Response).

Durante el proceso se intercambian los siguientes mensajes SAML2:

1. 1º AuthnRequest que el SP realiza para el intermediador Cl@ve.
2. 2º AuthnRequest que el intermediador Cl@ve prepara para el IdP.
3. 1º Response, la que el IdP realiza para el intermediador Cl@ve. El IdP responde los valores solicitados, firmados y cifrados (en el caso de que se requiera).
4. 2º Response, la que intermediador Cl@ve realiza al SP que inició la petición de autenticación.

Un SP sólo necesita tener en cuenta las peticiones de los puntos 1 y 2.

3.2. Peticiones SAML2

Las peticiones SAML2 (AuthnRequest) se inician desde un SP y son firmadas por éste. Estas peticiones se envían al nodo Clave mediante HTTP Post o HTTP Redirect, aunque se recomienda el tipo POST.

El nodo Clave, tras recibir a un usuario que porta un ticket SAML2 de autenticación, verifica la integridad/autenticidad de los mensajes de la petición. Esto implica los siguientes pasos:

1. Extraer el certificado de firma del SP y validarlo.
2. Validar la firma del mensaje SAML2 enviado en la petición.

Las peticiones que no se puedan validar de esta forma son rechazadas.

Posteriormente se extraerá la lista de IdPs habilitados y se continuará con el flujo de negocio normal para realizar la autenticación.

Puede consultar un ejemplo de petición comentada en el anexo **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**.

3.3. Respuestas SAML2

En el caso de las respuestas SAML2, éstas también se envían firmadas. Adicionalmente, la aserción que contiene el propio mensaje de respuesta, donde residen los datos personales del ciudadano, puede firmarse también.

Las aserciones SAML2 contienen información personal por lo que deben cifrarse empleando para ello el certificado obtenido de la request original, que es el empleado por el origen al que va dirigida la respuesta. Es decir, en cada paso de vuelta se descifran y vuelven a cifrar los datos personales del usuario. Es un cifrado adicional al cifrado que introduce el canal de comunicación SSL/TLS.

Por lo tanto, una vez identificado a un ciudadano, el IdP y posteriormente el intermediador responderán al SP con una respuesta SAML2 firmada y con los datos personales cifrados con el certificado indicado por el propio SP en sus ticket de solicitud.

Cada eslabón de la cadena debe verificar la integridad/autenticidad de los mensajes SAML2 antes de procesar la respuesta. Esto implica los siguientes pasos:

1) Extraer el certificado de firma y validarlo.
2) Validar la firma del mensaje SAML2 enviado en la respuesta.
3) Si la aserción SAML2 está firmada, su firma debe ser validada también.
4) Descifrar los datos personales contenidos en la respuesta. Sólo el nodo de destino puede hacerlo.

Las respuestas que no se puedan validar de esta forma serán rechazadas. Puede consultar un ejemplo de respuesta comentada en el anexo **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**.

3.4. Calidad de la credencial (LoA) y atributos personales

Un elemento esencial en el marco de interoperabilidad del reglamento eIDAS es el nivel de seguridad de la autenticación. El nivel de seguridad caracteriza el grado de confianza de un medio de identificación electrónica para establecer la identidad de una persona, garantizando así que la persona que afirma poseer una identidad determinada la tiene.

El nivel de seguridad depende del grado de confianza que aporte este medio de identificación electrónica sobre la identidad pretendida o declarada por una persona, teniendo en cuenta los procedimientos técnicos, (por ejemplo, prueba y verificación de la identidad, autenticación), las procedimientos de gestión (métodos utilizados por entidad para expidir la identidad electrónica) y los controles aplicados a todo el proceso.

El reglamento eIDAS establece los tres niveles de seguridad siguientes:

- el nivel de seguridad **bajo** se referirá a un medio de identificación electrónica, en el contexto de un sistema de identificación electrónica, que establece un **grado limitado de confianza** en la identidad pretendida o declarada de una persona y se describe en referencia a las especificaciones técnicas, las normas y los procedimientos del mismo, entre otros los controles técnicos, y cuyo objetivo es **reducir el riesgo de uso indebido o alteración de la identidad**;
- el nivel de seguridad **sustancial** se referirá a un medio de identificación electrónica, en el contexto de un sistema de identificación electrónica, que establece un **grado sustancial de confianza** en la identidad pretendida o declarada de una persona y se describe en referencia a las especificaciones técnicas, las normas y los procedimientos del mismo, entre otros los controles técnicos, y cuyo objetivo es **reducir sustancialmente el riesgo de uso indebido o alteración de la identidad**;
- el nivel de seguridad **alto** se referirá a un medio de identificación electrónica, en el contexto de un sistema de identificación electrónica, que establece un **grado de confianza** en la identidad pretendida o declarada de una persona **superior** al medio de identificación electrónica con un nivel de seguridad sustancial, y se describe en referencia a las especificaciones técnicas, las normas y los procedimientos del mismo, entre otros los controles técnicos, cuyo objetivo es **evitar el uso indebido o alteración de la identidad**.

Además, el reglamento eIDAS establece los siguientes datos a proporcionar para las **personas físicas**:

- Un conjunto de **3 datos obligatorios** que deben proporcionarse en todos los casos. Estos datos son:
  - **Identificador de unicidad**: Se trata de un identificador vinculado de manera única a una persona determinada, que permite asociar a la misma persona autenticaciones sucesivas. Se debe hacer notar que este identificador garantiza que no habrá dos personas con el mismo identificador, como es el caso del NIF español.
  - **Nombre** (en general uno o varios nombres)
  - **Apellido** (en general los apellidos del ciudadano)
- Un conjunto de **5 datos opcionales** que el emisor de la autenticación puede decidir proporcionar o no, en función de las características del esquema de identificación utilizado. El propósito de estos datos opcionales es facilitar la asociación de los datos de identificación del ciudadano en autenticaciones sucesivas. Estos datos opcionales son:
  - Fecha de nacimiento
  - Nombre al nacer
  - Apellido al nacer

- Lugar de nacimiento
- Dirección actual
- Género

4. Elementos que componen la arquitectura

4.1. Proveedor de Servicios (Service Provider o SP)

Es una aplicación (generalmente web), API o conjunto de aplicaciones que aportan la funcionalidad al usuario, a excepción del proceso de autenticación. En nuestro contexto se referirá a la aplicación de una entidad privada que ofrece servicios de cara al ciudadano.

Como SP, una entidad que interactúa con el nodo Cl@ve debe:

- Generar los tickets SAML2 de petición de autenticación que serán enviados al nodo Cl@ve.
- Validar tickets de respuesta SAML2 y descrifrarlas para obtener la información que se desea conocer del ciudadano.
- En el nuevo modelo, los SPs no necesitan preguntar por los atributos personales que desea conocer del usuario. En su lugar, envía una petición vacía y la plataforma devuelve todos los atributos que es posible enviar al SP.

4.2. Proveedor de identidad (Identity Provider o IdP)

De forma general, un IdP es el componente que almacena las identidades, y por tanto autentica e identifica a usuarios, atendiendo a las peticiones de datos que llegan desde los SPs pasando por Cl@ve o directamente.

La autenticación se obtiene lanzando un desafío al usuario que éste debe superar con algo que tiene, algo que sabe y/o algo que es.

4.3. Pasarela Cl@ve (proxy entre SPs e IdPs)

Es el elemento central que permite encaminar al usuario hacia el IdP adecuado, atendiendo a los requisitos del proveedor de servicio. Recibe al usuario mediante una redirección web, que porta un ticket SAML firmado por un SP, que es validado por el intermediador. El intermediador prepara a su vez un ticket similar y lanza la solicitud contra el IdP. Posteriormente recibe la respuesta SAML del IdP, la descifra y valida y procede a preparar un ticket de respuesta similar para el SP.

Al tratarse de elementos centrales se simplifica el establecimiento de la confianza, ya que tanto los proveedores de servicio como los proveedores de identidad sólo se comunican con la pasarela. Esta simplificación robustece el sistema y protege la credencial del usuario, que nunca sale de la relación directa con el IdP.

5. Proceso de alta y requisitos de acceso

El funcionamiento del sistema se basa en el establecimiento de un círculo de confianza entre el proveedor de servicios y la plataforma Cl@ve. Para ello será necesario que el proveedor de servicios obtenga un certificado para la firma de los tokens SAML2, cuya parte pública remitirá al gestor del nodo Cl@ve.

Esto supone que esos integradorestendrán que disponer de la capacidad para crear y procesar tickets SAML2. Para ello, se puede optar por dos estrategias de integración:

- Si no se dispone de un motor capaz de crear tickets SAML2, se proporcionan unas librerías de integración que incluyen el motor SAML2 y otras herramientas para integrarse. Estas utilidades, descritas en el presente documento, incluye el paquete de integración en Java, PHP y .Net, así como una plataforma de demo que se puede consultar como referencia, y contra la que integrar un SP localmente.
- Si se dispone de un motor SAML2, la integración puede hacerse directamente a través de intercambio de tickets SAML2 de acuerdo con el formato especificado.

Por su parte, el gestor del intermediador proporcionará al proveedor de servicio la parte pública del certificado usado para la firma de sus propios token SAML.

Adicionalmente, en este mismo documento se indican las URL, accesibles desde Internet, a las cuales se deben enviar las peticiones del entorno de pruebas, para que el proveedor de servicio pueda realizar todas las pruebas de conexión necesarias antes del paso a producción, cuya URL también se facilitará una vez que se haya determinado que la integración en el entorno de pruebas ha sido correcta.

6. Paquetes de integración para los SPs

Existen tres paquetes de integración distintos acordes a las tecnologías que los SPs dispongan (Java, ASP.NET y PHP). Se detallan a continuación los elementos que componen cada uno y su procedimiento de implantación.

En las redirecciones que se intercambian con la pasarela se deben realizar HTTP POST con los siguientes parámetros:

- RelayState: Es un parámtero alfanumérico externo que dificulta ataque s de repetición. Debe ser incluído en las peticiones y se devolverá en modo eco en las respuestas y en las peticiones de Logout con el mismo parámetro.
- SAMLRequest: Contiene el token SAML a enviar codificado en Base64.
- SAMLResponse: Contiene el token SAML de respuesta codificado en Base64.
- logoutRequest: Contiene el token SAML de logout a enviar codificado en Base64.
- logoutResponse: Contiene el token SAML de respuesta de logout codificado en Base64.

Un integrador debe utilizar el software de demo como un ejemplo funcional que permite utilizar la pasarela Cl@ve. Para una integración satisfactoria, es responsabilidad del SP integrar la funcionalidad en sus propios flujos de negocio, incluyendo las adaptaciones que sean necesarias. Los procesos necesarios para adaptar el software a las necesidades específicas del SP quedan fuera del alcance del presente paquete de integración.

6.1. Deshabilitar Proveedores de Identificación en la pasarela Cl@ve

El SP puede deshabilitar uno o varios IdPs para que no estén disponibles para la identificación del usuario.

En la imagen siguiente se ha deshabilitado la identificación mediante certificado.



Para deshabilitar un IdP es necesario enviar un atributo específico dentro de la petición SAML. Por ejemplo, para deshabilitar el IdP de certificados se envía lo siguiente con el valor isRequired=false

```
<eidas:RequestedAttribute FriendlyName="AFirmaIdP"
Name="http://es.minhafp.clave/AFirmaIdP"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
```

Los atributos disponibles para deshabilitar el IdP correspondiente son los siguientes:

- Un atributo personal de tipo “http://es.minhafp.clave/AFirmaIdP” para deshabilitar el IdP “Certificado electrónico” en la ventana de selección de la pasarela.
- Un atributo personal de tipo “http://es.minhafp.clave/GISSIdP” para deshabilitar el IdP “Clave permanente” en la ventana de selección de la pasarela.
- Un atributo personal de tipo “http://es.minhafp.clave/AEATIdP” para deshabilitar el IdP “PIN24H” en la ventana de selección de la pasarela.
- Un atributo personal de tipo “http://es.minhafp.clave/EIDASIdP” para deshabilitar el IdP “Credencial europea” en la ventana de selección de la pasarela.

6.2. Atributo RelayState en la petición SAML

Es posible incluir dentro de la petición SAML un atributo personal de tipo “http://es.minhafp.clave/RelayState” que será devuelto en el SAML de respuesta con el mismo valor que se ha enviado en la petición.

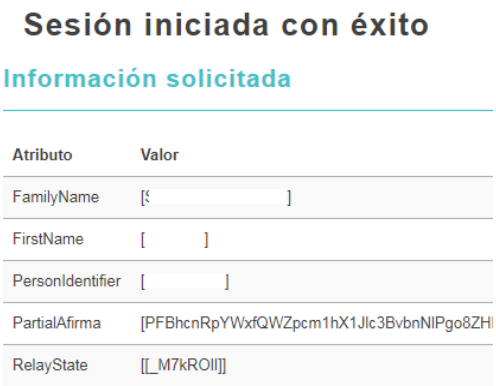
Es similar al parámetro POST que se intercambia por fuera del token SAML con la diferencia de que este se devuelve firmado por la pasarela.

Este atributo puede ser usado por los SP para cualquier tipo de comprobación que quieran realizar entre la petición y respuestas correspondientes o para persistir información en los ciclos asíncronos de autenticación.

6.3. Identificación mediante certificado

En caso de utilizar el proveedor de identidades preparado para certificados electrónicos, se obtiene una respuesta que incluye la siguiente información:

- Nombre del usuario
- Apellidos del usuario
- NIF o CIF del usuario.
- Respuesta parcial de @Firma con los mapeos de certificados.
- Campo RelayState interno que debe coincidir con el original.



Respuesta parcial de @firma

Con el objetivo de reducir el tamaño global del token SAML, en el atributo Partial Afirma se devuelven solamente dos nodos de toda la respuesta de @firma:

- El nodo **<dss:Result>** que contiene el resultado de la validación del certificado. Este nodo es redundante ya que es equivalente a la propia respuesta de Cl@ve. Pero se ha incluido porque en caso de resultar inválido, el ResultMinor puede resultar interesante para conocer la causa de la invalidez.
- El nodo **<afxp:ReadableCertificateInfo>** que contiene los mapeos propios de @firma y que puede utilizarse para obtener información adicional sobre el certificado:
  - ✓ Clasificación del certificado: persona física, persona jurídica, representante de las AA.PP. etc
  - ✓ Razón Social para determinados tipos de certificado
  - ✓ OID o política del certificado para detectar si es un DNI,
  - ✓ Emisor del certificado,
  - ✓ Etc

Los datos disponibles dentro del campo PartialAfirma están codificados en Base64.

Consultar más abajo un ejemplo de respuesta de Afirma devuelto por Cl@ve.

Certificados de persona física

Cl@ve solamente soporta certificados de persona física y certificados de persona jurídica anteriores al eIDAS y a extinguir.

Por tanto, Cl@ve solo devolverá respuesta válida para certificados con las siguientes clasificaciones.

- Clasificación = 0 – Persona física – certificado cualificado de firma
- Clasificación = 1 – Persona jurídica (no cualificado)
- Clasificación = 5 – Empleado Público
- Clasificación = 11 – Persona física representante ante las AAPP de persona jurídica
- Clasificación = 12 – Persona física representante ante las AAPP de entidad sin persona jurídica

Puede consultarse el documento “Tratamiento de Certificados en Afirma” que puede encontrarse en el PAe en la dirección https://administracionelectronica.gob.es/ctt/afirma/descargas.

Certificados emitidos por prestadores españoles

Cl@ve solamente soporta certificados emitidos por prestadores Españoles. Por tanto, Cl@ve solo devolverá respuesta válida para certificados con el valor “ES” en el nodo “país”.

Ejemplo de respuesta de @firma devuelto por Cl@ve en Base64:

```
<Partial_Afirma_Response>
    <dss:Result>
        <dss:ResultMajor>urn:oasis:names:tc:dss:1.0:resultmajor:Success</dss:ResultMajor>
```

```
        <dss:ResultMinor>urn:oasis:names:tc:dss:1.0:profiles:XSS:resultminor:valid:certificate:Definitive</ds
s:ResultMinor>
        <dss:ResultMessage xml:lang="es">El certificado es válido</dss:ResultMessage>
    </dss:Result>
        <afxp:ReadableCertificateInfo>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>OrganizacionEmisora</afxp:FieldIdentity>
                        <afxp:FieldValue>Agencia Notarial de Certificacion S.L.U. - CIF
B83395988</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>versionPolitica</afxp:FieldIdentity>
                        <afxp:FieldValue>23</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>usoCertificado</afxp:FieldIdentity>
                        <afxp:FieldValue>digitalSignature | nonRepudiation | keyEncipherment |
dataEncipherment</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>pais</afxp:FieldIdentity>
                        <afxp:FieldValue>ES</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>subject</afxp:FieldIdentity>
                        <afxp:FieldValue>CN=JUAN EJEMPLO
ESPAÑOL,serialNumber=11111111H,givenName=JUAN,SN=EJEMPLO ESPAÑOL,T=DIRECTOR,OU=Certificado Corporativo
Personal,OU=DEPARTAMENTO DE RRHH,O=EMPRESA CON CONVENIO CCP S.A.,C=ES</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>certQualified</afxp:FieldIdentity>
                        <afxp:FieldValue>YES</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>numeroSerie</afxp:FieldIdentity>
                        <afxp:FieldValue>1100659348503108952844565462754443472252</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>NombreApellidosResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>JUAN EJEMPLO ESPAÑOL</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>idPolitica</afxp:FieldIdentity>
                        <afxp:FieldValue>MITyC</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>tipoCertificado</afxp:FieldIdentity>
                        <afxp:FieldValue>ANCERT PF Corporativos Personales Firma 2010 SW</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>certClassification</afxp:FieldIdentity>
                        <afxp:FieldValue>ESIG</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>clasificacion</afxp:FieldIdentity>
                        <afxp:FieldValue>0</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>NIFResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>11111111H</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>nombreResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>JUAN</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>unidadOrganizativa</afxp:FieldIdentity>
                        <afxp:FieldValue> OrganizationUnit=DEPARTAMENTO DE RRHH OrganizationUnit=Certificado
Corporativo Personal</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>ApellidosResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>EJEMPLO ESPAÑOL</afxp:FieldValue>
                </afxp:ReadableField>
```

```
                <afxp:ReadableField>
                        <afxp:FieldIdentity>segundoApellidoResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>ESPAÑOL</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>organizacion</afxp:FieldIdentity>
                        <afxp:FieldValue>EMPRESA CON CONVENIO CCP S.A.</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>primerApellidoResponsable</afxp:FieldIdentity>
                        <afxp:FieldValue>EJEMPLO</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>validoHasta</afxp:FieldIdentity>
                        <afxp:FieldValue>2019-05-04 sáb 10:12:17 +0200</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>validoDesde</afxp:FieldIdentity>
                        <afxp:FieldValue>2016-05-04 mié 10:12:17 +0200</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>email</afxp:FieldIdentity>
                        <afxp:FieldValue>no-reply@ancert.com</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>qscd</afxp:FieldIdentity>
                        <afxp:FieldValue>UNKNOWN</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>idEmisor</afxp:FieldIdentity>
                        <afxp:FieldValue>CN=ANCERT Corporativos Personales V2,O=Agencia Notarial de
Certificacion S.L.U. - CIF B83395988,L=Paseo del General Martinez Campos 46 6a planta 28010
Madrid,C=ES</afxp:FieldValue>
                </afxp:ReadableField>
                <afxp:ReadableField>
                        <afxp:FieldIdentity>politica</afxp:FieldIdentity>
                        <afxp:FieldValue>1.3.6.1.4.1.18920.2.1.1.2.4</afxp:FieldValue>
                </afxp:ReadableField>
        </afxp:ReadableCertificateInfo>
</Partial_Afirma_Response>
```

6.4. Paquete de integración Java

Este paquete está asociado a la integración de aplicaciones J2EE y permitirá realizar ciclos de autenticación.

Requisitos del sistema mínimo para un correcto funcionamiento de la solución:

Table 1 Matriz de requisitos

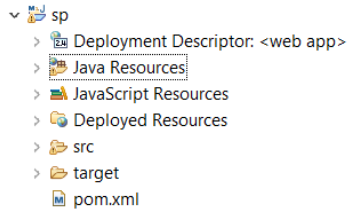
Concepto	Valores testados	Valores soportados
Maven	2.7	2.X 3.X
Versión de Java	1.8	1.7 o superior
Servidores de aplicaciones	Wildfly 10.1.Final Apache Tomcat 7.X	Wildfly 10.1.Final Apache Tomcat 7.X Apache Tomcat 8.X Glassfish 4.X Weblogic 12C
IDE	Eclipse	Eclipse Netbeans InetlliJ IDEA JDeveloper 11G

Otros requisitos requeridos para la implantación:

- Disponer de un certificado válido en formato PKCS#12 (extensiones *.p12 o *.pfx) y PKCS#8 (extensiones *.pem y *.der), expedido por una CA reconocida.
- Para entornos preproductivos este certificado puede ser auto-firmado. Dado que el SP debe firmar los mensajes XMLs el certificado debe soportar esta funcionalidad.

La solución se basa en Maven, por tanto las librerías y resto de clases necesarias estarán incluidas en las dependencias descritas en el fichero "***pom.xml***". En el paquete de integración se proveen los proyectos compilados de los artefactos que no están publicadas en los repositorios centrales de Maven.

Accediendo desde un IDE de desarrollo, debería de visualizarse algo similar a:



6.4.1. Estrategias de integración

Despliegue desde cero

Se entregan unas librerías de Cl@ve v2.0, ya importadas en un ejemplo funcional, compiladas con Java 1.8 y preparadas como artefactos Maven.

- Con este ejemplo funcional se entregan diferentes Servlet de Spring para preparar una SAMLRequest y recibir una SAMLResponse utilizando las anteriores librerías.
- Es un ejemplo completo que no incluye ningún mecanismo de sesión más allá de mostrar los datos recibidos.
- Este ejemplo funcional consiste en un flujo web simple que en última instancia llama a la clase "ReturnAction.java".

Migracion desde Cl@ve 1.0

Si lo que se desea es cambiar librerías en un SP prexistente, ya integrado con Cl@ve 1.0, el propio ejemplo constituye una referencia comentada sobre los nuevos objetos a manejar y los métodos que invocar para seguir el protocolo Cl@ve. En esta clase se indica:

- La instancia del motor SAML requiere unos ficheros de configuración nuevos, pero muy similares al modelo Stork del anterior paquete de integración, con un fichero SamlEngine.xml como fichero de configuración maestro que invoca a un fichero SignModule para el acceso al JKS de firma y validación y a un fichero de atributos soportados llamado "saml-engine-eidas-attributes.xml". Se instancia de forma similar a Stork.
- Dicha instancia pasa de ser "STORKSAMLEngine" a ser una instancia de la interfaz "ProtocolEngineNoMetadataI".
- Que para obtener el objeto que constituye una SAMLRequest legítima se deja de utilizar "STORKAuthnRequest" y se pasa a utilizar "IAuthenticationRequestNoMetadata".
- Que para manejar un objeto que constituye una SAMLResponse a enviar se deja de utilizar "STORKAuthnResponse" y se pasa a utilizar "IResponseMessageNoMetadata" .
- Que para validar una request SAML se deja de invocar el método "engine.validateSTORKAuthnRequest(byte[])" y se pasa a invocar el método "engine.unmarshallRequestAndValidate(byte[])".
- Que para firmar y cifrar una response SAML se deja de invocar el método "engine.generateSTORKAuthnResponse" y se pasa a invocar el método "engine.generateResponseMessage".
- El nuevo motor cifra la aserción que contiene los atributos personales del usuario. Es configurable firmar esta aserción siendo su valor por defecto, true.
- La lista de atributos disponibles utilizan nuevos constructores y deben ajustarse a un esquema descrito en un XML. Se provee de un ejemplo de inyección de atributos.

Pasos para la migración

El kit de integración incluye el documento *"Migración del kit de Clave1 a Clave2.docx"* que detalla pormenorizadamente los cambios que hay que realizar al kit de Cl@ve 1.0 para migrarlo a Cl@ve 2.0 y que funcione en este nuevo entorno. También se incluyen los diferentes parches git correspondientes a cada commit al realizar la migración.

Se puede usar como referencia para entender cómo adaptar a Cl@ve 2.0 las aplicaciones que fueron desplegadas para conectarse a Cl@ve 1.0 y que se basaron en el kit de integración de ese entorno.

Discontinuación del kit de Cl@ve 1.0

Es importante recordar que con la publicación del paquete de integración de Cl@ve 2.0 se discontinúa y se deja de dar soporte al paquete de Cl@ve 1.0. Por tanto, si decide adaptarse un desarrollo basado en el kit de Cl@ve 1.0 será responsabilidad del organismo mantener actualizadas las librerías correspondientes. El kit de Cl@ve 1.0 estaba basado en Struts mientras que el de Cl@ve 2.0 está basado en Spring.

6.4.2. Ficheros de propiedades

La ruta hacia la carpeta en la cual residen los ficheros de configuración se declara en el fichero interno de propiedades ubicado en "SP\src\main\resources\spEnvironmentContext.xml".

El fichero de propiedades "sp.properties" contiene información de configuración como es el nombre del proveedor, la URL del módulo, la URL a la que se retornará la respuesta de la pasarela Cl@ve (después de realizar la petición SAML).

El fichero "***saml-engine-additional-attributes.xml***" permite incluir nuevos atributos soportados al esquema. No todos los atributos pueden ser recabados, ya que hace falta un IdP que pueda proveerlos.

6.4.3. Descripción de los ficheros de configuración de un SP

Para configurar el sistema se utilizan ficheros de configuración. Para definir las rutas de base donde buscar dichos ficheros se debe proveer la variable como un parámetro de sistema o un parámetro de Java:

- `CLAVE_SP2_CONFIG_REPOSITORY`.- Ruta a la configuración del módulo. Por ejemplo {ruta}/Config/SP/

La implementación del motor presente en el módulo de integración actúa de una forma específica, permitiendo la escalabilidad de este aspecto. Por ejemplo, el sistema ya incluye una implementación para poder obtener datos de configuración:

- De un fichero maestro donde se apunta a los otros ficheros.
- Ficheros donde se define la configuración para firma. Suelen apuntar a un JKS.
- Ficheros donde se define la configuración para la capa de cifrado.
- Ficheros donde se definen aspectos del protocolo, como por ejemplo, las validaciones habilitadas.
- Ficheros donde se define el set de atributos para atributos adicionales.
- Ficheros de tipo sp.properties donde se configuran URLS y que contienen el ProviderName empleado o una lista blanca de algoritmos de firma y cifrado.

Por lo tanto, existe una carpeta "***config***" se encuentra todos los ficheros de configuraciones que permiten ajustar el SP al sistema en el que vaya a implantarse.

Los cambios a realizar en los ficheros serán los siguientes:

SignModule_SP.xml:

- La etiqueta *<entry key="**keystorePath**">* debe tener la ruta en la que se vaya a guardar el almacén de certificados (donde estará el certificado privado de SP para realizar firmas y también los certificados públicos en los que se confía).
- La etiqueta <entry key="**keyStorePassword**"> hace referencia a la contraseña del almacén.
- La etiqueta <entry key="**keyPassword**"> hace referencia a la contraseña del certificado que se desea usar.
- La etiqueta <entry key="**issuer**"> hace referencia a los datos que deberá tener el certificado.
- La etiqueta <entry key="**serialNumber**"> hace referencia al número de serie del certificado.

sp.properties:

- **provider.name:** Nombre identificativo de la aplicación a tres niveles de profundidad, es decir, indicando un nombre de institución, un sector dentro de la institución y una aplicación dentro del sector. Este nombre debe ser consensuado durante el proceso de alta.
- **sp.aplication**: Identifica a una aplicación como subconjunto dentro del SP.
- **service.url:** URL del servicio Cl@ve.
- **sp.return**: dirección en la que se espera la respuesta procedente de la pasarela Clave.

EncryptModule_SP.xml:

- La etiqueta *<entry key="**keystorePath**">* ha de cambiarse por la dirección hacia el donde esté el almacén de certificados.
- La etiqueta <entry key="**keyStorePassword**"> hace referencia a la contraseña del almacén.
- La etiqueta <entry key="**keyPassword**"> hace referencia a la contraseña del certificado que se desea usar.
- La etiqueta <entry key="**responseDecryptionIssuer**"> hace referencia a los datos del certificado del SP.
- La etiqueta <entry key="**serialNumber**"> hace referencia al número de serie del certificado.

En resumen, para una integración solo es necesario tomar en cuenta los ficheros "sp.properties", "EncryptModule_SP.xml" y "SignModule_SP.xml".

6.4.4. Preparación del entorno de desarrollo

Las dependencias del proyecto están gestionadas mediante Maven. Por ello, existe un fichero descriptivo donde se indican los siguientes datos:

pom.xml:

- **artifactId**: Identificador con el que se reconocerá al SP. En función de este parámetro se generará el WAR, aunque es posible cambiar el nombre posteriormente antes de desplegarlo en el servidor de aplicaciones.
- **name**: nombre que recibe el propio SP.
- **parent-version**: version del SP.

- **Description**: Descripción del propio SP.

Las dependencias que no están provistas en los repositorios centrales de Maven se proveen en la carpeta "Dependencias". Dentro de esta carpeta se entrega el pom padre y las librerías compiladas y mavenizadas.

Los jar están compilados empleando Java 1.8. En caso de que sea necesario compilarlos con otra versión de Java, se provee también el código fuente de los proyectos.

6.4.5. Certificados

La comunicación del módulo SP se basa en certificados electrónicos. Por lo tanto la pasarela Cl@ve tiene que confiar en el certificado del SP y éste tiene que confiar en el certificado de la pasarela. El intercambio de certificados se produce durante un proceso de alta y registro previo.

En el fichero "SignModule_SP.xml" se configura la ruta del almacén de certificados, el cual tiene que contener el certificado privado que usará SP para realizar firmas, así como los certificados públicos en los que se confía.

En el fichero "EncryptModule_SP.xml" se configura la ruta del almacén de certificados, el cual tiene que contener el certificado privado que usará SP para realizar descifrados.

Requisitos en el apartado de certificados

Es necesario que el JKS disponga del certificado propio del servidor.

Dado que un SP debe comunicarse con la pasarela, la disponibilidad de este certificado en el contendor de certificados es un requisito para su correcta configuración.

Limitaciones de los certificados

Ha de destacarse que **no** podrá haber dos certificados en el almacén de certificados que contentan el mismo ***issuer y serial number*.**

Consideraciones adicionales

Los tickets SAML2 valen para un único ciclo. No deben reutilizarse. Si es necesario enviar una nueva solicitud, esta debe ser calculada de nuevo.

El sistema se basa en redirecciones del usuario que porta un ticket SAML2 firmado por la entidad que redirecciona y que debe validar la entidad que recibe la visita. Por lo tanto, sólo es necesario que cada elemento del sistema sea visible por parte del usuario, excluyendo la necesidad de que los servidores sean visibles entre sí.

6.4.6. Dependencias a importar

- Dependencias necesarias para generar un ticket SAML:
  - `eu.eidas.eidas-commons`

    - eu.eidas.eidas-saml-engine

- Dependencias de las anteriores librerías que no están en el repositorio central de maven:
    - eu.eidas.eidas-encryption
    - eu.eidas.eidas-light-commons
    - eu.eidas.eidas.configmodule
    - org.apache.velocity.velocity-1.7-eidas_1
    - org.opensaml.opensaml-2.6.5-eidas_1
    - org.opensaml.openws-1.5.5
    - org.opensaml.xmltooling-1.4.5

- Dependencias de las anteriores librerías
    - org.bouncycastle.bcprov-jdk15on
    - org.apache.santuario.xmlsec

- Dependencias del Proyecto web que no son necesarias para la lógica de negocio, sólo para los Servlets que se proveen como ejemplo:
    - javax.servlet.servlet-api
    - javax.servlet.jsp-api
    - org.springframework.spring-web

El proyecto de ejemplo se provee con un fichero "pom.xml" para organizar las dependencias mediante Maven. Al final del fichero se ha declarado un repositorio local para dejar automáticamente disponibles para Maven las dependencias en los BuildingBlocks de eIDAS v1.4.

Aquellas dependencias que no se encuentran en el repositorio central de Maven se proveen en el propio paquete de integración, en la carpeta /fuentes/sp/Dependencias, incluyendo su fichero POM. No obstante, es posible descargar el motor SAML de los Building Blocks de eIDAS disponibles en:

- https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS-Node+-+All+releases

6.4.7. Exposición de servicios

La funcionalidad del módulo actuando como SP implica la necesidad de servir una URL que se corresponde con un servlet. Se parte de la premisa de que ya existe una página web en el SP donde se deberá incluir un botón que comience el proceso de autenticación, por lo que se requiere una URL donde recibir la respuesta.

Durante un ciclo de autenticación se espera una respuesta de forma asíncrona, por lo que se expone una esta URL donde se espera recibir un ticket SAMLResponse firmado por la pasarela . Esta respuesta SAML tiene un atributo llamado "InResponseTo" que contiene el mismo valor que el atributo ID de la petición. De esta manera, se puede enlazar la respuesta con su origen.

Para servir esta URL, como se puede observar en el fichero "WEB-INF/web.xml", en el apartado "Servlet definition", se define un endpoint en "https://<dominio>/ReturnPage" que ejecuta el "doGet" y "doPost" de la clase "es.clave.sp.ReturnAction". Este método valida la respuesta SAML, la descifra y da como resultado una página JSP que contine los atributos recibidos. Esta funcionalidad se comenta más adelante. A continuación se puede observar la declaración del servlet utilizando Struts2.

```
<servlet>
   <description>ReturnPage</description>
   <display-name>ReturnPageServlet</display-name>
   <servlet-name>ReturnPageServlet</servlet-name>
   <servlet-class>es.clave.sp.actions.ReturnAction</servlet-class>

</servlet>
```

Posteriormente, se asocial el end-point con el servlet

```
<servlet-mapping>
        <servlet-name>ReturnPageServlet</servlet-name>
        <url-pattern>/ReturnPage</url-pattern>
</servlet-mapping>
```

En el ejemplo hay otros servlets adicionales para:

- PopulateIndexPage: Popular la página de inicio
- IndexPage: Firmar una petición SAML y enviarla a la pasarela.
- Capturar errores.

Una vez que se ha configurado la solicitud a enviar, un usuario pulsa sobre el botón enviar. Este botón envía el formulario a un servlet llamado "IndexPage". Como se puede observar en el fichero "WEB-INF/web.xml", se define un endpoint en "https://<dominio>/IndexPage" que ejecuta el "doPost" de la clase "es.clave.sp.IndexAction". Éste método prepara una solicitud de autenticación SAML2 con el formulario recibido y escribe el resultado en el JSP llamado "redirect.jsp" que hace una redirección vía HTTP POST hacia la URL de la pasarela.

6.4.8. Generación de un ticket SAML2

Véase el método “doGet” de la clase “sp.clave.sp.IndexAction”. Para crear un ticket SAML2 empleando el motor SAML2 de eIDAS se requieren las siguientes llamadas:

En primer lugar, es necesario instanciar el motor SAML2. Para ello se ha de invocar estáticamente a la factoría incluida en el paquete de integración “SpProtocolEngineFactory” indicando como parámetro el nombre del módulo en cuestión. En este caso, el módulo se ha llamado SP.:

```
ProtocolEngineNoMetadataI protocolEngine = SpProtocolEngineFactory.getSpProtocolEngine(“SPNoMetadata”);
```

Para levantar la instancia del motor, en primer lugar se va a cargar el fichero “SPSamlEngine.xml”, y se va a buscar una instancia con el identificador indicado:

- <instance name=" SPNoMetadata">

A partir de este bloque de configuración se obtienen los nombres de los otros ficheros de configuración del motor SAML necesarios para la firma y el cifrado. En el paquete de integración estos ficheros de configuración se llaman “SignModule_SP.xml” y “EncryptModule_SP.xml”.

Una vez que se dispone de una instancia del motor SAML2 se puede empezar a utilizar para diferentes cosas. Para solicitar la creación de una nueva SAMLRequest, primero hay que construir un objeto de tipo “EidasAuthenticationRequestNoMetadata” e introducir la configuración que se desea:

```
// Se genera la estructura de datos para configurar una nueva solicitud SAML
EidasAuthenticationRequestNoMetadata.Builder reqBuilder = new
EidasAuthenticationRequestNoMetadata.Builder();
// Se indica la URL de destino (hacia dónde va dirigido el ticket)
reqBuilder.destination(nodeUrl);
// Se indica el id del SP, obtenido durante el proceso de alta. Será el
DIR3_NIF;SPApplication
reqBuilder.providerName(providerName);
// Se incluye el array de atributos personales que se desean conocer
reqBuilder.requestedAttributes(reqAttrMap);
// Se indica el nivel mínimo de la credencial que puede utilizar el usuario
reqBuilder.levelOfAssuranceComparison(LevelOfAssuranceComparison.fromString("minimum").strin
gValue());
reqBuilder.levelOfAssurance(LevelOfAssurance.LOW.stringValue());
// Se indica el sector al que pertenece el SP, en clave siempre será public
reqBuilder.spType(SpType.PUBLIC.toString());
// Se calcula un ID-token aleatorio. No debe haber dos tickets distintos con el mismo ID.
reqBuilder.id(SAMLEngineUtils.generateNCName());
```

Una vez que se dispone de una solicitud configurada, se ha de proceder a firmarla y enviarla. Para ello se invoca al método siguiente de la instancia SAML2 disponible:

```
// Se firma la petición indicando los datos del destino
IRequestMessageNoMetadata bytes =
protocolEngine.generateRequestMessage(reqBuilder.build(), true);
// Se extraen los bytes
byte[] ticket = bytes.getMessageBytes();
```

```
// Se codifican en Base64 para su envío
String samlRequest = EidasStringUtil.encodeToBase64(ticket);
```

6.4.9. Validación de una respuesta SAML2

Véase el método " doPost" de la clase "es.clave.sp.ReturnAction". Para validar un ticket SAML2 empleando el motor SAML se requiere el siguiente código.

En primer lugar, es necesario instanciar el motor SAML2. Para ello se ha de invocar estáticamente a la factoría incluida en el paquete de integración "SpProtocolEngineFactory" indicando como parámetro el nombre del módulo en cuestión. En este caso, el módulo se ha llamado SP:

```
ProtocolEngineNoMetadataI protocolEngine = SpProtocolEngineFactory.getSpProtocolEngine("SPNoMetadata");
```

Para levantar la instancia del motor, en primer lugar se va a cargar el fichero "SPSamlEngine.xml", y se va a buscar una instancia con el identificador indicado:

```
<instance name="SPNoMetadata">
```

El código para validar el ticket es el siguiente.

```
// Primero se extraen los bytes de la respuesta recibida
byte[] decSamlTicket = EidasStringUtil.decodeBytesFromBase64(SAMLResponse);
// Se lee y se valida
IAuthenticationResponseNoMetadata authnResponse;
try {
    // En caso de error durante la validación se produce una excepción
    authnResponse = engine.unmarshallResponseAndValidate(decSamlTicket,
request.getRemoteHost(), 0, 0, configs.getProperty(Constants.SP_RETURN ));
    // En caso de que se haya recibido una respuesta cifrada, se puede descifrar
inependientemente
    if (SPUtil.isEncryptedSamlResponse(decSamlTicket)) {
       byte[] eidasTicketSAML = engine.checkAndDecryptResponse(decSamlTicket);
       samlUnencryptedResponseXML =
SPUtil.extractAssertionAsString(EidasStringUtil.toString(eidasTicketSAML));
    }
} catch (EIDASSAMLEngineException e) {
    logger.error(e.getMessage(), e);
}
// Con el objeto obtenido se puede comprobar si es una autenticción exitosa y
extraer los datos recibidos.
if (authnResponse.isFailure()) {
    throw new ApplicationSpecificServiceException("Saml Response is fail",
authnResponse.getStatusMessage());
} else {
    attrMap = authnResponse.getAttributes().getAttributeMap();
}
```

6.4.9.1. Atributos recibidos en la respuesta

Los atributos personales que es posible recibir de la pasarela, que identifican al usuario serán los siguientes:

- http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName.- Contiene el nombre del usuario. Se puede recibir tanto un nombre simple como uno compuesto. Es un atributo personal obligatorio que se recibirá siempre.
- http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName.- Contiene los apellidos del usuario. Es un atributo personal obligatorio que se recibirá siempre.
- http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier.- Contiene el identificador único (NIF) del Usuario. Es un atributo personal obligatorio que se recibirá siempre.
- http://es.minhafp.clave/RelayState.- Contiene el mismo valor enviado en la petición.

En el caso de identificación con certificado también se devuelve el siguiente atributo:

- http://es.minhafp.clave/PartialAfirma.- Contiene la respuesta parcial de @firma en Base64.

6.4.10. Generación de un ticket SAML2 de petición de Logout

Véase el método “generateLogout” de la clase “es.clave.sp.ReturnAction”. Para generar un ticket de logout empleando el motor SAML2 se requieren las siguientes llamadas:

En primer lugar, es necesario instanciar el motor SAML2. Para ello se ha de invocar estáticamente a la factoría incluida en el paquete de integración “SPUtil.getProtocolEngine()”.

```
defaultProtocolEngineFactory = new
ProtocolEngineFactoryNoMetadata(protocolEngineConfigurationFactory)
```

Para levantar la instancia del motor, en primer lugar se va a cargar el fichero “IdPSamlEngine.xml”, y se va a buscar una instancia con el identificador indicado:

```
<instance name="SPNoMetadata">
```

Una vez que se dispone de una instancia del motor SAML2 se puede empezar a utilizar para diferentes cosas. Para solicitar la validación de una LogoutRequest, primero hay que construir un objeto de tipo “LogoutRequest” e introducir la configuración que se desea:

```
LogoutRequest authnRequest = null;
protocolEngine.generateLogoutRequestMessage(assertionConsumerUrl,
providerName, destination, nonce)
```

Una vez que se dispone de una solicitud creada, se puede codificar en Base64 y proceder a su envío a la pasarela.

6.4.11. Validación de una respuesta SAML2 de logout

Véase el método “doPost” de la clase “es.clave.sp.ReturnAction”. Para validar un ticket SAML2 empleando el motor SAML se requiere el siguiente código.

En primer lugar, es necesario instanciar el motor SAML2. Para ello se ha de invocar estáticamente a la factoría incluida en el paquete de integración

```
ProtocolEngineNoMetadataI protocolEngine =
        SpProtocolEngineFactory.getSpProtocolEngine(Constants.SP_CONF);
```

Para levantar la instancia del motor, en primer lugar se va a cargar el fichero “IdPSamlEngine.xml”, y se va a buscar una instancia con el identificador indicado:

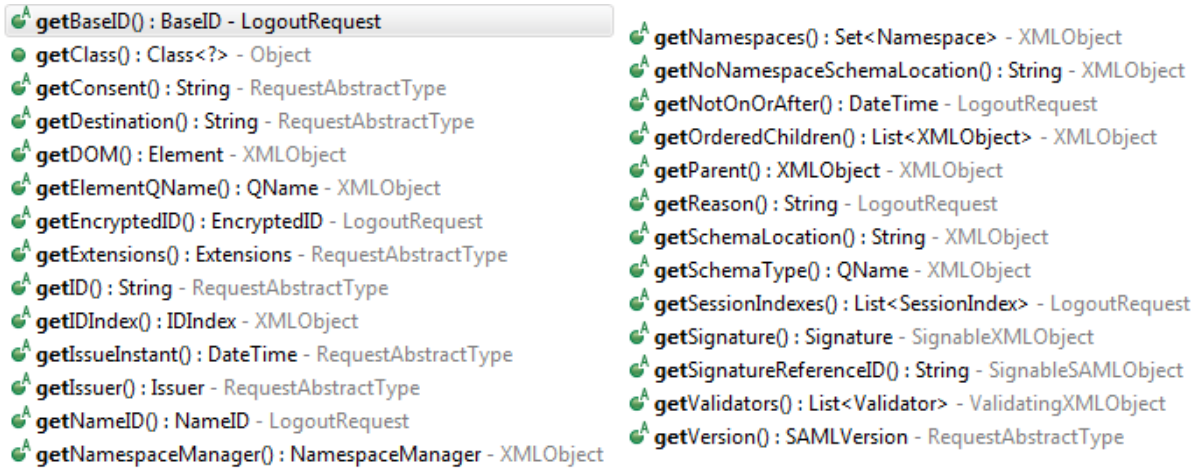
```
<instance name="SPNoMetadata">
```

El código para validar el ticket es el siguiente.

```
LogoutResponse logoutResp = null;
try {
      byte[] decSamlToken = EidasStringUtil.decodeBytesFromBase64(logoutResponse);
```

```
        //validate SAML Token
        logoutResp = protocolEngine.unmarshallLogoutResponseAndValidate(decSamlToken,
                                        request.getRemoteHost(), 0, 0,
configs.getProperty(Constants.SP_RETURN));
} catch (Exception e) {
        Logger.error("No se pudo validar la respuesta", e);
}
// Check session
String requestSamlId = SessionHolder.sessionsSAML.get(relayState);
if (requestSamlId == null || !requestSamlId.equals(logoutResp.getInResponseTo())) {
        throw new InvalidParameterEIDASException("La respuesta recibida no corresponde
con ninguna request o no coincide el RelayState");
}
```

Estos son algunos de los datos diponibles



6.4.12. Guía de integración de SP

Un SP puede integrarse fácilmente con cualquier servidor descrito en **¡Error! No se encuentra el origen de la referencia.**.

A modo de ejemplo y por facilidad de entendimiento usaremos un servidor Apache Tomcat v7.0.53 para la realización del procedimiento básico de configuración. Los pasos a realizar son:

- Instalación del certificado del SP e importación del mismo en el JKS, incluyendo la clave privada.

```
keytool -import -alias tomcat -keystore <your_keystore_filename> -file
<your_certificate_filename>
```

6.4.13. Información adicional

Para extraer URLs de servicio y extraer el certificado en base al entorno a configurar:

- Servicios estables.
- Producción

Al igual que en el procedimiento anterior lo incluimos en el JKS.

```
keytool -import -alias tomcat -keystore <your_keystore_filename> -file
<your_certificate_filename>
```

- Configuración del puerto SSL del servidor de aplicaciones:

En nuestro caso de ejemplo se ha de modificar el fichero *server.xml* que se encuentra en la carpeta ***conf*** *del servidor de aplicaciones*.

El puerto a *configurar* es el 443. Los cambios más importantes son:

- Habilitar tráfico SSL sin autenticación de cliente.
- Indicar los ficheros de certificados correspondientes para el handshake SSL (hay que indicar el certificado con el que se identifica el servidor).
- El protocolo que maneja el tráfico entrante es *HTTP/1.1* y el esquema es HTTPS.

Para un servidor Tomcat, la configuración de los puertos quedaría más o menos así:

```
<Connector SSLEnabled="true" clientAuth="false" connectionTimeout="20000" keystoreFile=[…]
keystorePass=[…] keystoreType="JKS" port="443" protocol="HTTP/1.1" scheme="https"
secure="true" />
```

- Modificar los ficheros:
  - "pom.xml": con la información del servicio o despliegue "artifactId", y demás valores que requiere el fichero.
  - "SignModule_SP.xml", incluyendo los datos referentes a los certificados.
  - "EncryptModule_SP.xml": con los datos referentes a los certificados.
- Desplegar el fichero .war en la carpeta "*deployments"* correspondiente (en el caso de Tomcat, en "*/webapps"*) y arrancar el servidor.

PARA UNA SOLUCIÓN EXISTENTE QUE SE QUIERE INTEGRAR SE REQUIERE:

- Disponer en el CLASS_PATH del servidor de aplicaciones los paquetes definidos en Dependencias a importar
- Incluir en el web.xml de la aplicación a integrar los servlets de acceso como se indica en Exposición de servicios.
- Hay que importar las funcionalidades definidas en Generación de un ticket SAML y Validación de una respuesta SAML.
- La aplicación a integrar tan solo debe incluir en su interfaz de usuario un único botón que inicie el proceso de autenticación. No es necesario importar la funcionalidad que hay en el ejemplo, que permite a un usuario configurar la SAMLRequest. En un escenario real, el propio SP es quien decide la configuración del ticket.

Proceso de verificación:

Se podrá comprobar el correcto funcionamiento del sistema (una vez se hayan configurado los parámetros anteriores), accediendo desde un navegador al nombre cualificado de su servicio: https://FQDN/SP, o en el caso de que disponga de un proxy inverso, compruébelo directamente accediendo al servidor y solicitando

las página para localhost, https://localhost/SP2 (siempre que su servidor de aplicaciones escuche en este interfaz).

NOTA: El sistema se quejará del certificado, ya que no está expedido para localhost y es un certificado autofirmado sin valor como evidencia.

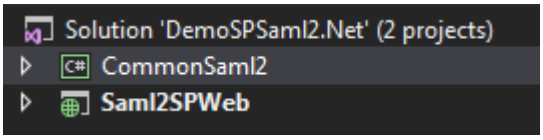
Verifique por favor la salida de logs de su servidor de aplicaciones, en busca de posibles errores de configuración.

6.5. Paquete de integración C# y ASP.NET

Los requisitos ténicos requeridos para poder implementar la integración para la tecnología .NET son:

- .Net Framework 4.6.1. Disponer de una implementación que soporte el algoritmo de firma RSA-SHA512.
- Un IDE para el desarrollo. Por ejemplo, Visual Studio 2017. El IDE incluye un IISExpress para probar el despliegue.

Para comenzar, hay que abrir la solución del proyecto en: ***source\repos\DemoSPSaml2\DemoSPSaml2.Net.sln***.



CommonSaml2 es una librería que expone un API estático con los métodos necesarios para la lógica del programa (métodos de cifrado, cálculo de firmas, etc.).

Saml2SPWeb contiene la parte Web, fundamentalmente el flujo Web para realizar un ciclo de autenticación completo (desde que el usuario se introduce en la página web en su navegador hasta que recibe toda la información desde el IdP). Se trata de una implementación de ejemplo, siendo necesario que el proveedor de servicios lo adecue a su impementación propia. **Nota**: Se ha hecho uso del IIS Express y NuGET para el uso de este proyecto.

En este proyecto existen tres apartados destacados:

1. **SP.aspx**: Vista y código de cómo se genera un ticket (permite cambiar la forma de autenticación, el valor del LoA…).
2. **ResponseHandler.cs**: Código que valida la respuesta de la pasarela Cl@ve y redirige al usuario, si todo ha ido bien, a SuccesPage.aspx.
3. SuccesPage.aspx: tiene como finalidad mostrar los datos propios del usuario y permitir al usuario desloguearse emitiendo otro ticket SAML2, el cual si todo ha ido bien, le llevará a la vista SP.aspx.

Se debe comprobar que se está usando la versión de Framework 4.6.1 en todos los proyectos, para que no aparezcan incompatibilidades en el desarrollo, para ello se ha de hacer *click derecho* sobre las soluciones, *propiedades/aplicación/plataforma de destino.*

6.5.1. Ficheros de propiedades

Para la demo se ha creado un manejador de la configuración llamado *CommonSaml2/Conf.cs,* el cual contiene las definiciones de los parámetros de configuración. Los valores de éstos parámetros de configuración se encuentran en "Saml2SPWeb/APP_DATA/properties.xml".

Este fichero es el lugar donde un integrador debe centrar su atención, pues permite indicar los parámetros de funcionamiento:

```
<properties
    url_clave = "https://se-pasarela-clave.gob.es/Proxy2/ServiceProvider"
    url_redirect_method = "post"
    provider_name = "DEMO-SP"
    dir_p12_signing = "~/App_Data/Firma.p12"
    cert_password_sign = "changeit"
    dir_log = "eidas-error-log.txt"
    WantAssertionsSigned = "false"
  >
```

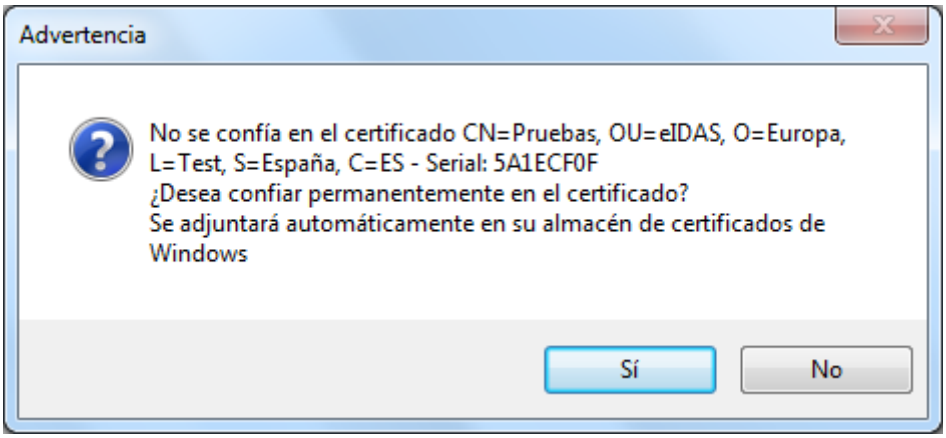
6.5.2. Certificados

Es requisito disponer de un certificado X509 con un par de claves que se usará para el descifrado asimétrico y para las tareas de firma en formato PKCS12 (.p12 ó .pfx).

En el fichero properties.xml se encuentran los parámetros que contienen el path hacia estos ficheros disponibles en "Saml2SPWeb/App_Data":

```
    dir_p12_signing = "~/App_Data/Firma.p12"
    cert_password_sign = "changeit"
```

Cabe destacar, que los certificados de confianza serán almacenados en el almacén de certificados propio de Windows dedicado a personas de confianza, sólo disponibles para el usuario actual. En caso de que no se confíe en un certificado, se mostrará un mensaje de advertencia que permitirá incluir el certificado de forma permanente.



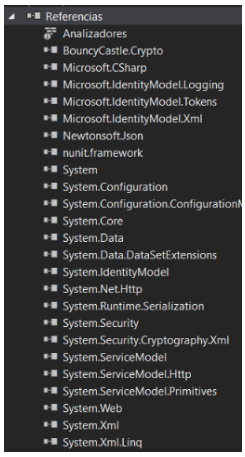
6.5.3. Dependencias a importar

En cada solución hay un apartado denominado referencias. Ahí se encuentras todas las librerías del proyecto.

Las dependencias se administran mediante el sistema ***Nuget***, y la forma de hacerlo es la siguiente:

Para descargar las dependencias, en Visual Studio se puede lanzar una compilación (Proyecto/compilar) o se puede hacer clic con el botón secundario en cada proyecto. En este caso se abrirá un menú contextual en el cual se puede escoger la opción *Administrar paquetes nuget...* Posteriormente, se abrirá una ventana y en la opción de *Examinar* habrá un buscador. Tras esta tarea, se puede pulsar en el icono de descargar. Esto instalará las dependendencias de forma automática en el proyecto seleccionado.

Se debe incluir en este destino las siguientes librerías o descargarlas en el caso que no se disponga ya de las mismas en el apatardo de referencias.



6.5.4. Generación de un ticket SAML

Acceda a http(s)://dominio:puerto/SP.aspx. Se mostrará una ventana con dos opciones.



Hay dos formas de crear un ticket saml:

Se puede crear uno predeterminado dando en el botón enviar, en el cual te da todas las opciones disponibles para la autenticación del usuario, y otro personalizado de manera que es el mismo usuario el que elije cómo desea autenticarse etc

El método se encuentra en DemoSPSaml2.Net/ CommonSaml2/ XMLData.cs y son **Saml2RequestDefault.**

Saml2RequestDefault recibe una serie de parámetros que provienen de Conf.cs. Si en la vista SP.aspx se cambiara alguno, serían sustituidos del ticket SAML los usados por defector por los estos segundos.

El parámetro que puede ser modificado es newLOA (para cambiar el valor de loa por low, substancial o high). El resto se usan como Flag para enviar o no un atributo en el ticket SAML2. Estos atributos son todos los Boolean.

El array values y el string urlService deben de salir de la configuración, ya que siempre han de ser los mismos.

Code 1 Saml2RequestDefault

```
       public static XmlDocument Saml2RequestDefault(string[] values, string urlService, Boolean afirma, Boolean clavePermanente, Boolean pin24h, Boolean eidas, Boolean auth, Boolean relaystate, string newLOA)
        {
            XmlDocument doc = new XmlDocument();
            doc.PreserveWhitespace = false;

            try
            {
                XmlNode docNode = doc.CreateXmlDeclaration("1.0", "UTF-8", null);
                doc.AppendChild(docNode);
                XmlNode samlreq = doc.CreateElement("saml2p", "AuthnRequest", "urn:oasis:names:tc:SAML:2.0:protocol");

                XmlAttribute destination = doc.CreateAttribute("Destination");
                destination.Value = urlService;
                samlreq.Attributes.Append(destination);
                //Adding ProviderName
                XmlAttribute providerName = doc.CreateAttribute("ProviderName");
                providerName.Value = Conf.get_provider_name();
                samlreq.Attributes.Append(providerName);
                //Adding a random ID
                XmlAttribute id = doc.CreateAttribute("ID");
                id.Value = "_" + Guid.NewGuid().ToString();
                samlreq.Attributes.Append(id);
                //Adding the hour and date
                XmlAttribute issueInstant = doc.CreateAttribute("IssueInstant");
                issueInstant.Value = Utils.GetDate();
                samlreq.Attributes.Append(issueInstant);
                //Adding SP return URL
                XmlAttribute assertionConsumerURL = doc.CreateAttribute("AssertionConsumerServiceURL");
                assertionConsumerURL.Value = Conf.get_url_return_response();
                samlreq.Attributes.Append(assertionConsumerURL);

                XmlAttribute attribute0 = doc.CreateAttribute("xmlns", "saml2p", "http://www.w3.org/2000/xmlns/");
                attribute0.Value = "urn:oasis:names:tc:SAML:2.0:protocol";
```

```
                XmlAttribute     attribute1     =     doc.CreateAttribute("xmlns",     "ds",
"http://www.w3.org/2000/xmlns/");
                attribute1.Value = "http://www.w3.org/2000/09/xmldsig#";

                XmlAttribute    attribute2    =    doc.CreateAttribute("xmlns",    "eidas",
"http://www.w3.org/2000/xmlns/");
                attribute2.Value = "http://eidas.europa.eu/saml-extensions";

                XmlAttribute    attribute3    =    doc.CreateAttribute("xmlns",    "saml2",
"http://www.w3.org/2000/xmlns/");
                attribute3.Value = "urn:oasis:names:tc:SAML:2.0:assertion";

                XmlAttribute attribute4 = doc.CreateAttribute("Consent");
                attribute4.Value = "urn:oasis:names:tc:SAML:2.0:consent:unspecified";

                XmlAttribute attribute6 = doc.CreateAttribute("ForceAuthn");
                if (auth) attribute6.Value = "true";
                else attribute6.Value = values[0];

                XmlAttribute attribute8 = doc.CreateAttribute("IsPassive");
                attribute8.Value = values[1];

                XmlAttribute attribute11 = doc.CreateAttribute("Version");
                attribute11.Value = "2.0";

                samlreq.Attributes.Append(attribute0);
                samlreq.Attributes.Append(attribute1);
                samlreq.Attributes.Append(attribute2);
                samlreq.Attributes.Append(attribute3);
                samlreq.Attributes.Append(attribute4);
                samlreq.Attributes.Append(attribute6);
                samlreq.Attributes.Append(attribute8);
                samlreq.Attributes.Append(attribute11);

                XmlNode    NameIDPolicy    =    doc.CreateElement("saml2p",    "NameIDPolicy",
"urn:oasis:names:tc:SAML:2.0:protocol");
                XmlAttribute AllowCreate = doc.CreateAttribute("AllowCreate");
                AllowCreate.Value = values[2];
                NameIDPolicy.Attributes.Append(AllowCreate);
                XmlAttribute Format = doc.CreateAttribute("Format");
                Format.Value = "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified";
                NameIDPolicy.Attributes.Append(Format);

                XmlNode     extensions     =     doc.CreateElement("saml2p",     "Extensions",
"urn:oasis:names:tc:SAML:2.0:protocol");

                XmlNode       sptype       =       doc.CreateElement("eidas",       "SPType",
"http://eidas.europa.eu/saml-extensions");
                sptype.AppendChild(doc.CreateTextNode("public"));
                extensions.AppendChild(sptype);

                XmlNode    req    =    doc.CreateElement("eidas",    "RequestedAttributes",
"http://eidas.europa.eu/saml-extensions");
                extensions.AppendChild(req);
```

```
                if (afirma)
                {
                    XmlNode    re0    =    doc.CreateElement("eidas",    "RequestedAttribute",
"http://eidas.europa.eu/saml-extensions");
                    XmlAttribute atre00 = doc.CreateAttribute("FriendlyName");
                    atre00.Value = "AFirmaIdP";

                    XmlAttribute atre01 = doc.CreateAttribute("Name");
                    atre01.Value = "http://es.minhafp.clave/AFirmaIdP";

                    XmlAttribute atre02 = doc.CreateAttribute("NameFormat");
                    atre02.Value = "urn:oasis:names:tc:SAML:2.0:attrname-format:uri";

                    XmlAttribute atre03 = doc.CreateAttribute("isRequired");
                    atre03.Value = "false";

                    re0.Attributes.Append(atre00);
                    re0.Attributes.Append(atre01);
                    re0.Attributes.Append(atre02);
                    re0.Attributes.Append(atre03);

                    req.AppendChild(re0);
                }
                if (clavePermanente)
                {
                    XmlNode    re1    =    doc.CreateElement("eidas",    "RequestedAttribute",
"http://eidas.europa.eu/saml-extensions");
                    XmlAttribute atre10 = doc.CreateAttribute("FriendlyName");
                    atre10.Value = "GISSIdP";

                    XmlAttribute atre11 = doc.CreateAttribute("Name");
                    atre11.Value = "http://es.minhafp.clave/GISSIdP";

                    XmlAttribute atre12 = doc.CreateAttribute("NameFormat");
                    atre12.Value = "urn:oasis:names:tc:SAML:2.0:attrname-format:uri";

                    XmlAttribute atre13 = doc.CreateAttribute("isRequired");
                    atre13.Value = "false";

                    re1.Attributes.Append(atre10);
                    re1.Attributes.Append(atre11);
                    re1.Attributes.Append(atre12);
                    re1.Attributes.Append(atre13);

                    req.AppendChild(re1);
                }
                if (pin24h)
                {
                    XmlNode    re2    =    doc.CreateElement("eidas",    "RequestedAttribute",
"http://eidas.europa.eu/saml-extensions");
                    XmlAttribute atre20 = doc.CreateAttribute("FriendlyName");
                    atre20.Value = "AEATIdP";
```

```
                    XmlAttribute atre21 = doc.CreateAttribute("Name");
                    atre21.Value = "http://es.minhafp.clave/AEATIdP";

                    XmlAttribute atre22 = doc.CreateAttribute("NameFormat");
                    atre22.Value = "urn:oasis:names:tc:SAML:2.0:attrname-format:uri";

                    XmlAttribute atre23 = doc.CreateAttribute("isRequired");
                    atre23.Value = "false";

                    re2.Attributes.Append(atre20);
                    re2.Attributes.Append(atre21);
                    re2.Attributes.Append(atre22);
                    re2.Attributes.Append(atre23);

                    req.AppendChild(re2);
                }
                if (eidas)
                {
                    XmlNode    re3    =    doc.CreateElement("eidas",    "RequestedAttribute",
"http://eidas.europa.eu/saml-extensions");
                    XmlAttribute atre30 = doc.CreateAttribute("FriendlyName");
                    atre30.Value = "EIDASIdP";

                    XmlAttribute atre31 = doc.CreateAttribute("Name");
                    atre31.Value = "http://es.minhafp.clave/EIDASIdP";

                    XmlAttribute atre32 = doc.CreateAttribute("NameFormat");
                    atre32.Value = "urn:oasis:names:tc:SAML:2.0:attrname-format:uri";

                    XmlAttribute atre33 = doc.CreateAttribute("isRequired");
                    atre33.Value = "false";

                    re3.Attributes.Append(atre30);
                    re3.Attributes.Append(atre31);
                    re3.Attributes.Append(atre32);
                    re3.Attributes.Append(atre33);

                    req.AppendChild(re3);
                }
                if (relaystate)
                {
                    XmlNode    re4    =    doc.CreateElement("eidas",    "RequestedAttribute",
"http://eidas.europa.eu/saml-extensions");
                    XmlAttribute atre40 = doc.CreateAttribute("FriendlyName");
                    atre40.Value = "RelayState";

                    XmlAttribute atre41 = doc.CreateAttribute("Name");
                    atre41.Value = "http://es.minhafp.clave/RelayState";

                    XmlAttribute atre42 = doc.CreateAttribute("NameFormat");
                    atre42.Value = "urn:oasis:names:tc:SAML:2.0:attrname-format:uri";

                    XmlAttribute atre43 = doc.CreateAttribute("isRequired");
                    atre43.Value = "false";
```

```
                re4.Attributes.Append(atre40);
                re4.Attributes.Append(atre41);
                re4.Attributes.Append(atre42);
                re4.Attributes.Append(atre43);

                req.AppendChild(re4);
            }

            XmlNode       requestAuthnContext       =       doc.CreateElement("saml2p",
"RequestedAuthnContext", "urn:oasis:names:tc:SAML:2.0:protocol");
            XmlAttribute comparison = doc.CreateAttribute("Comparison");
            comparison.Value = "minimum";
            requestAuthnContext.Attributes.Append(comparison);

            XmlNode       authcontextclassref       =       doc.CreateElement("saml2",
"AuthnContextClassRef", "urn:oasis:names:tc:SAML:2.0:assertion");
            //Adding LoA
            if(newLOA                                !=                                null)
authcontextclassref.AppendChild(doc.CreateTextNode(newLOA));
            else
authcontextclassref.AppendChild(doc.CreateTextNode(Conf.get_loa_forms()[0]));

            requestAuthnContext.AppendChild(authcontextclassref);

            samlreq.AppendChild(extensions);
            samlreq.AppendChild(NameIDPolicy);
            samlreq.AppendChild(requestAuthnContext);
            //END
            doc.AppendChild(samlreq);
        }
        catch (Exception e)
        {
            //Log: "Error creating the request. Any Conf data could be null: " + e
        }

        return doc;
    }
```

6.5.5. Validación de una respuesta

En este caso, se debe recoger la respuesta enviada por parte de la pasarela. Esta respuesta XML contendrá aserciones que han de ser inspeccionadas para comprobar que el servidor ha validado los datos que le hemos proporcionado, y que nos ha mandado los datos correctamente. El ejemplo usado para esta demo se encuentra en Saml2SPWeb/ResponseHandler.cs.

El orden de validación es:

- Recoger el nodo “saml2p:statusMessage” y comprobar que es success.
- Comprobar que el certificado está en nuestro key store de certificados de confianza
- Comprobar la firma.

- Comprobar que la respuesta coincide con la petición.
- Si es necesario, descifrar los datos personales.

```
     //There is the process to get the response from Cl@ve node
                string samlResponseField =
ConfigurationSettingsHelper.GetCriticalConfigSetting("samlResponseField");
                string responseData = context.Request.Form[samlResponseField];

                byte[] reqDataB64 = Convert.FromBase64String(responseData);
                Stream stream = new MemoryStream(reqDataB64);

                //Convert the response to XML form
                XmlDocument xml = new XmlDocument();
                xml.PreserveWhitespace = true;
                xml.Load(stream);
                //Getting the issuer node
                var issuer = xml.GetElementsByTagName("saml2:Issuer");
                string urlMetadata = issuer[0].InnerText;
                //Getting metadata from url to validate
                XmlDocument xmlMetadata = XMLData.ReadMetadata(urlMetadata);
                bool metadataVerified = Utils.VerifyMetadata(xmlMetadata, false);


                //checks Cl@veNode metadata and SAML response. true as parameter saves cert
in your key store
                if (metadataVerified && Saml2Code.Utils.VerifyResponse(xml))
                {
                    string status =
xml.GetElementsByTagName("saml2p:StatusMessage")[0].InnerText;

                    //checking if request was success in the post method
                    if (status == "urn:oasis:names:tc:SAML:2.0:status:Success")
                    {
                        XmlDocument request = (XmlDocument)context.Session["SAMLRequest"];
                        string id =
request.FirstChild.Attributes.GetNamedItem("ID").InnerText;
                        string inResponseTo =
xml.FirstChild.Attributes.GetNamedItem("InResponseTo").InnerText;
                        if (id != null && inResponseTo != null && id.Equals(inResponseTo))
                        {
                            XmlDocument xml_response = null;
                            if (xml.GetElementsByTagName("saml2:EncryptedAssertion").Count >
0)
                            {
                                var dataDecripted = CipherUtils.Decrypt(xml);
                                xml_response = new XmlDocument();
                                xml_response.LoadXml(dataDecripted);
                            }
                            else
                            {
                                xml_response = xml;
                            }

                            context.Session["SAMLResponse"] = xml_response;
                            context.Session["result"] = "success";
                            //Redirect to the next
                            returnAction = Conf.get_url_response_success();
                        } else
                        {
```

```
                            Utils.DoLog(new Exception("InResponseTo field does not match
Request ID"));
                        }
                    }
                    else
                    {
                        Utils.DoLog(new Exception("Is a SAMLFail response: " + status));
                    }
                }
                else
                {
                    Utils.DoLog(new Exception("Response or metadata NOT valid"));
                }
            }
            catch (Exception ex)
            {
                Utils.DoLog(ex);
                returnAction = Conf.get_url_return_if_fail_metadata_eidas_node();
            }
            context.Response.Redirect(returnAction);
```

6.5.6. Descifrado de una respuesta

En primer lugar se debe de realizar el descifrado asimétrico. El resultado del mismo, nos permitirá obtener una clave, la cual usaremos en el descifrado simétrico. Los métodos que se muestran a continuación se encuentran en CommonSaml2/CipherUtils.cs.

Para el primer descifrado se debe de utilizar nuestro propio certificado en formato pem (recordemos que todo se obtiene del fichero de configuración CommonSaml2/Conf.cs).

6.5.6.1. Descifrado asimétrico

```
public static byte[] AsymmetricDecrypt(XmlDocument xml)
        {
       //We must get the cipherValue to decrypt
            string data = xml.GetElementsByTagName("xenc:CipherValue")[0].InnerText;

            IAsymmetricBlockCipher cipher = new OaepEncoding(new RsaEngine(), new
Sha1Digest());

            TextReader reader;

            AsymmetricCipherKeyPair key;
       //We must use our .pem with private and public keys
            using (reader = File.OpenText(@Conf.get_dir_pem()))
            {
                key = (AsymmetricCipherKeyPair)new PemReader(reader).ReadObject();
            }

            byte[] cipheredBytes = Convert.FromBase64String(data);

            RsaKeyParameters privateKey = (RsaKeyParameters)key.Private;

            cipher.Init(false, privateKey);

            byte[] deciphered = cipher.ProcessBlock(cipheredBytes, 0, cipheredBytes.Length);
       //The method will return decripted cipherValue (the symmetric key)
            return deciphered;
        }
```

6.5.6.2. Descifrado simétrico

```
private static byte[] DecryptWithKey(byte[] encryptedMessage, byte[] key,
            int nonSecretPayloadLength = 0)
        {
//We know that gcm encrypt will have a authentication bytes
            if (key == null || key.Length != 256 / 8)
            {
                throw new ArgumentException(String.Format("Key needs to be {0} bit!", 256),
"key");
            }

            if (encryptedMessage == null || encryptedMessage.Length == 0)
            {
                throw new ArgumentException("Encrypted Message Required!",
"encryptedMessage");
            }

            using (var cipherStream = new MemoryStream(encryptedMessage))
            using (var cipherReader = new BinaryReader(cipherStream))
            {
                var nonce = cipherReader.ReadBytes(96 / 8);
```

```
                var cipher = new GcmBlockCipher(new AesFastEngine());
                var parameters = new AeadParameters(new KeyParameter(key), 128, nonce,
null);
                cipher.Init(false, parameters);

                var cipherText = cipherReader.ReadBytes(encryptedMessage.Length -
nonSecretPayloadLength - nonce.Length);
                var plainText = new byte[cipher.GetOutputSize(cipherText.Length)];

                try
                {
                    var len = cipher.ProcessBytes(cipherText, 0, cipherText.Length,
plainText, 0);
                    cipher.DoFinal(plainText, len);
                }
                catch (InvalidCipherTextException)
                {
                    //Return null if it doesn't authenticate
                    return null;
                }
//the text with decripted attributes from user
                return plainText;
            }
        }
```

El siguiente método se encargará de hacer uso de los anteriores, simplificando el número de llamadas.

```
public static string Decrypt(XmlDocument xml, int nonSecretPayloadLength = 0)
        {
              //Asymmetric decrypt
            var key = AsymmetricDecrypt(xml);

            byte[] data =
Convert.FromBase64String(xml.GetElementsByTagName("xenc:CipherValue")[1].InnerText);
       //Symmetric decrypt
            var plaintext = DecryptWithKey(data, key, nonSecretPayloadLength);

            return Encoding.ASCII.GetString(plaintext);

        }
```

6.5.7. Log out

Para realizar el Log out es tan sencillo como presionar en la vista final en la que se muestran los atributos del usuario el botón “Log out”. Enviará un ticket SAML2 con la petición de desloguearse y recibirá la confirmación. Una vez recibida, devolverá al usuario a la primera vista SP.aspx.

El código está en XMLData.cs / saml2LogOut().

```
public static XmlDocument saml2LogOut()
        {
            XmlDocument doc = new XmlDocument();
            doc.PreserveWhitespace = false;
            try
            {
                // logoutrequest/BEGIN
```

```
                XmlNode logoutrequest = doc.CreateElement("saml2p", "LogoutRequest",
"urn:oasis:names:tc:SAML:2.0:protocol");

                XmlAttribute attribute0 = doc.CreateAttribute("xmlns", "saml2p",
"http://www.w3.org/2000/xmlns/");
                attribute0.Value = "urn:oasis:names:tc:SAML:2.0:protocol";

                XmlAttribute attribute1 = doc.CreateAttribute("xmlns", "ds",
"http://www.w3.org/2000/xmlns/");
                attribute1.Value = "http://www.w3.org/2000/09/xmldsig#";

                XmlAttribute attribute2 = doc.CreateAttribute("xmlns", "eidas",
"http://www.w3.org/2000/xmlns/");
                attribute2.Value = "http://eidas.europa.eu/saml-extensions";

                XmlAttribute attribute3 = doc.CreateAttribute("xmlns", "saml2",
"http://www.w3.org/2000/xmlns/");
                attribute3.Value = "urn:oasis:names:tc:SAML:2.0:assertion";

                XmlAttribute Destination = doc.CreateAttribute("Destination");
                Destination.Value = Conf.get_url_eidas();

                XmlAttribute ID = doc.CreateAttribute("ID");
                ID.Value = "_" + Guid.NewGuid().ToString();

                XmlAttribute IssueInstant = doc.CreateAttribute("IssueInstant");
                IssueInstant.Value = Conf.get_valid_until();

                XmlAttribute NotOnOrAfter = doc.CreateAttribute("NotOnOrAfter");
                NotOnOrAfter.Value = Conf.get_valid_until();

                XmlAttribute Reason = doc.CreateAttribute("Reason");
                Reason.Value = "urn:oasis:names:tc:SAML:2.0:logout:user";

                XmlAttribute Version = doc.CreateAttribute("Version");
                Version.Value = "2.0";

                logoutrequest.Attributes.Append(attribute0);
                logoutrequest.Attributes.Append(attribute1);
                logoutrequest.Attributes.Append(attribute2);
                logoutrequest.Attributes.Append(attribute3);
                logoutrequest.Attributes.Append(Destination);
                logoutrequest.Attributes.Append(ID);
                logoutrequest.Attributes.Append(IssueInstant);
                logoutrequest.Attributes.Append(NotOnOrAfter);
                logoutrequest.Attributes.Append(Reason);
                logoutrequest.Attributes.Append(Version);
                //logoutrequest/END

                // issuer/BEGIN
                XmlNode issuer = doc.CreateElement("saml2", "Issuer",
"urn:oasis:names:tc:SAML:2.0:assertion");
                issuer.AppendChild(doc.CreateTextNode(Conf.get_url_return_logout()));
                logoutrequest.AppendChild(issuer);
                //issuer/END

                // nameid/BEGIN
                XmlNode nameid = doc.CreateElement("saml2", "NameID",
"urn:oasis:names:tc:SAML:2.0:assertion");

                XmlAttribute Format = doc.CreateAttribute("Format");
                Format.Value = "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified";
```

```
            XmlAttribute SPNameQualifier = doc.CreateAttribute("SPNameQualifier");
            SPNameQualifier.Value = Conf.get_url_return_response();

            nameid.Attributes.Append(Format);
            nameid.Attributes.Append(SPNameQualifier);

            nameid.AppendChild(doc.CreateTextNode(Conf.get_provider_name()));

            logoutrequest.AppendChild(nameid);
            // nameid/END

            doc.AppendChild(logoutrequest);
            return doc;
        }
        catch (Exception e)
        {
            //TODO
        }
        return null;
    }
```

6.6. Paquete de integración PHP

Este paquete de integración está destinado a SPs que deseen hacer uso de la tecnología PHP. Para su correcto funcionamiento, el sistema ha de cumplir una serie de requisitos mínimos:

Table 3 Matriz de requisitos PHP

Concepto	Valores testados	Valores soportados
Versión de PHP	5.6.31	5.6.31 o superiores
Servidores de aplicaciones	Apache 2.4.26	Apache Web Server
IDE	Netbeans 8.2	Netbeans 8.2 o superiores

6.6.1. Ficheros de configuración

Para poder adaptar el paquete a cada SP es necesario realizar algunas modificaciones en los ficheros de configuración.

En primer lugar, dentro de la carpeta *Config* se encuentra ***Authsources.php***. En este fichero debemos indicar en primer lugar un identificador de nuestro SP y a partir de ahí, definir el valor de una serie de parámetros relativos al mismo:

```
$config = array(
//Identificador de nuestro SP, se utilizará posteriormente para acceder a la configuración
       'demo-sp-php' => array(
              'saml:SP',
//Nombre del fichero del certificado de nuestro Proveedor de Servicios. Con él, este SP
podrá recibir respuestas encriptadas de la pasarela Cl@ve y firmará y validará todos los
mensajes.
              'certificate' => 'Stork.crt',
//Nombre del fichero del certificado del nodo eIDAS
              'validate.certificate' => 'certificadoProxy.crt',
//Nombre del fichero con la clave privada del SP. Se utilizará para firmar y desencriptar
los mensajes recibidos de la pasarela Cl@ve.
              'privatekey' => 'Stork.key',
//Contraseña de la clave privada del certificado del SP.
              'privatekey_pass' => 'admin',
              'name' => array(
                     'en' => 'demo-sp-php',
                     'pt' => 'demo-sp-php',
                     'es' => 'demo-sp-php',
              ),
//Nombre del formato usado en los atributos
       'attributes.NameFormat' => 'urn:oasis:names:tc:SAML:2.0:attrname-format:uri',
//Habilitar la firma del AuthnRequest
              'sign.authnrequest' => TRUE,
//URL de nuestro proveedor de servicio
        'entityID' => 'http://localhost/SP',
//URL del IdP Proxy
        'idp' => 'http://localhost:8080/Proxy2/ServiceProvider',
        'idp_binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
//URL del endpoint del SP que procesará las peticiones de Logout
        'logout.url' => 'http://localhost/SP/logoutResponse.php'
```

En segundo lugar, se deberán cambiar dos variables dentro del directorio lib/SAML2 en el fichero *Constants.php*:

```
//ID del SP en uso, necesario para recibir credenciales alemanas
const SPID = 'demo-sp-php';
const SPAPPLICATION = 'SPApp';

//URL del endpoint del SP que gestionará las respuestas de la pasarela Cl@ve.
const ASSERTION_URL = ' http://localhost/SP/return.php';
```

En el formulario inicial, el usuario podrá seleccionar cuál de los IdPs deshabilitar. En caso de no seleccionar ninguno, todos los IdP aparecerán habilitados por defecto. En este fichero se declaran los IdP disponibles:

```
public static $idsAttrs = array('AFirmaIdP', 'GISSIdP', 'AEATIdP', 'EIDASIdP');
```

Este array contiene todos los IdPs soportados. Para agregar un nuevo IdP, se debe añadir una nueva entrada en el mismo y su correspondiente entrada en $*attrs*:

```
public static $attrs = array(
//Nombre del nuevo IdP
            'AFirmaIdP.name' => 'AFirmaIdP',
//URI del nuevo IdP
             'AFirmaIdP.uri' => 'http://es.minhafp.clave/AFirmaIdP',
//Formato del espacio de nombres del nuevo IdP
              'AFirmaIdP.nameFormat' => 'urn:oasis:names:tc:SAML:2.0:attrname-format:uri',
//Valor por defecto del nuevo IdP
              'AFirmaIdP.value' => NULL,
```

6.6.2. Certificados

Como se ha comentado anteriormente, la confianza en la comunicación del módulo SP y la pasarela Cl@ve eIDAS se basa en certificados electrónicos. Por lo tanto, la pasarela tiene que confiar en el certificado del SP y éste tiene que confiar en el certificado de la pasarela. Para poder establecer esta relación de confianza, tenemos que tener en cuenta lo siguiente;

- En el fichero *config/config.php* se indicará cual será el directorio en el que se almacenarán los diferentes certificados, tanto los utilizados para firmar, como los empleados para validar respuestas (certificado de la pasarela Cl@ve de confianza) y descifrar contenido recibido.

```
$config = array('certdir' => 'cert/',
```

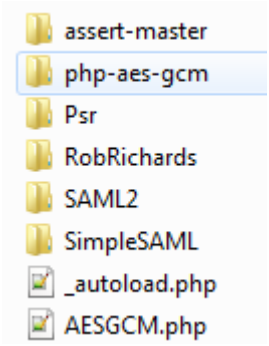
Por lo tanto, en el almacén de certificados */cert* deberán estar:

- ✓ 'certificate' => 'Stork.crt' – Certificado que el SP presentará en sus metadatos y que el nodo Cl@ve deberá almacenar para establecer confianza entre ambos.
- ✓ 'validate.certificate' => 'certificadoProxy.crt' – Certificado de la pasarela Cl@ve que nuestro Proveedor de Servicios almacenará para verificar posteriormente que las respuestas obtenidas provienen de un nodo Cl@ve de confianza.

- ✓ 'privatekey' => 'Stork.key' – Clave privada con la que nuestro Proveedor de Servicios firmará y desencriptará las aserciones.

6.6.3. Librerías utilizadas

En la carpeta */lib* se encuentran todas las librerías utilizadas en este paquete:



En este caso, el paquete de integración PHP se ha basado en el proyecto SimpleSAMLphp de código abierto, por ello se puede ver la existencia de librerías propias del proyecto (SAML2, SimpleSAML).

Adicionalmente se han utilizado librerías externas como:

- *assert-master*, para la extracción y validación de las aserciones
- *php-aes-gcm*, para la integración de criptografía de clave asimétrica con el algoritmo aes-gcm.
- *Psr,* para generar mensajes de log utilizados para la captura y visualización de errores
- *RobRichads*, para el uso de criptografía en ficheros XML.

6.6.4. Recogida de datos previa de las fuentes de autenticación

- **Constants**: Clase en la que se encuentran definidas las principales constantes utilizadas en todo el paquete y los métodos para recoger los atributos que serán solicitados en la petición. En primer lugar, será utilizada para obtener el ID de nuestro SP y de nuestra application.
  - *genDefaultAttrs* – Metodo utilizado para recoger las opciones seleccionadas por el usuario.

- **SimpleSAML_Auth_Source:** Clase básica encargada de la autenticación del usuario y de la obtención de las fuentes de autenticación del mismo. Mediante el método *getById*, a partir de nuestro ID y de la configuración existente en el fichero *authsources.php*, devuelve un objeto de la clase *sspmod_saml_Auth_Source_SP*.

```
$authSource = SAML2\Constants::SPID;
$as = new SimpleSAML_Auth_Simple($authSource);
$as = SimpleSAML_Auth_Source::getById($authSource);

//Se carga la configuración de nuestro Proveedor de Servicio de config/authsources.php
$localConfig = $as->getLocalConfig();
//Se carga de la configuración la url del IdP Proxy.
$idp[0]= $as->getidp();
```

```
$idp[1]= $as->getidp_binding();

//Se carga la información del IdP de la configuración
$idplocalConfig = $as->getIdPfromConfig($idp);
$session = SimpleSAML_Session::getSessionFromRequest();
$session->cleanup();

try {
  $state['saml:sp:AuthId'] = $authSource;

//Cargar extensiones.
       $extensions = Constants::genDefaultAttrs($_POST);
```

6.6.5. Construcción de la petición de autenticación

Clases utilizadas en la construcción de la petición de autenticación:

- **eidas_saml_Message**: Subclase que extiende de la clase sspmod_saml_Message que define el código para la construcción de mensajes en formato SAML2 basados en la configuración disponible.

```
    $ar = eidas_saml_Message::buildAuthnRequest($extensions, $forceauthn, $localConfig,
$idplocalConfig);
```

  - *BuildAuthnRequest* – Método que construye la petición de autenticación basándose tanto en la información de la configuración como en los atributos previamente seleccionados. Para ello se basa en objetos de la subclase *SAML2_EidasAuthnRequest* que extienden de la clase *AuthnRequest*. A partir de este objeto comienza a añadir nodos con los parámetros correspondientes

```
public static function buildAuthnRequest($extensions, $forceauthn, SimpleSAML_Configuration
$splocalConfig, SimpleSAML_Configuration $idplocalConfig) {

              $ar = new SAML2\SAML2_EidasAuthnRequest();

              if ($spMetadata->hasValue('NameIDPolicy')) {
                     $nameIdPolicy = $spMetadata->getString('NameIDPolicy', NULL);
              } else {
                     $nameIdPolicy = $spMetadata->getString('NameIDFormat',
SAML2\Constants::NAMEID_TRANSIENT);
              }

              if ($nameIdPolicy !== NULL) {
                     $ar->setNameIdPolicy(array(
                            'Format' => $nameIdPolicy,
                            'AllowCreate' => TRUE,
                     ));
              }
                /**
                 * Look for the endpoints in the information retrieved
                 */
              $dst = $idplocalConfig->getDefaultEndpoint('SingleSignOnService',
array(SAML2\Constants::BINDING_HTTP_REDIRECT));
              $dst = $dst['Location'];
```

```
                /**
                 * Start building the message for the authentication request
                 */
              $ar->setIssuer($splocalConfig->getString('entityid'));
              $ar->setDestination($dst);
              $ar->setForceAuthn($splocalConfig->getBoolean('ForceAuthn', $forceauthn));
              $ar->setIsPassive($splocalConfig->getBoolean('IsPassive', FALSE));

              $protbind = $splocalConfig->getValueValidate('ProtocolBinding', array(
                           SAML2\Constants::BINDING_HTTP_POST,
                           SAML2\Constants::BINDING_HTTP_ARTIFACT,
                           SAML2\Constants::BINDING_HTTP_REDIRECT,
                     ), SAML2\Constants::BINDING_HTTP_POST);

              /* Setting the appropriate binding based on parameter in sp configuration
defaults to HTTP_POST */
              $ar->setProtocolBinding($protbind);

              if ($spMetadata->hasValue('AuthnContextClassRef')) {
                     $accr = $splocalConfig->getArrayizeString('AuthnContextClassRef');
                     $ar->setRequestedAuthnContext(array('AuthnContextClassRef' => $accr));
              }

             if (!empty($extensions)) {
               $ar->setExtensions($extensions);
             }

              self::addRedirectSign($splocalConfig, $idplocalConfig, $ar);
              return $ar;
       }
```

- *addRedirectSign* – Método que incluye la clave de firma y el certificado del emisor (SP) al mensaje. Sus parámetros de entrada son la configuración del emisor, la configuración existente relativa al destinatario y el mensaje.

```
private static function addRedirectSign(SimpleSAML_Configuration $splocalConfig,
SimpleSAML_Configuration $idplocalConfig, SAML2\SAML2_EidasAuthnRequest $message)
  {

    if ($message instanceof SAML2_LogoutRequest || $message instanceof SAML2_LogoutResponse)
{
      $signingEnabled = $splocalConfig->getBoolean('sign.logout', NULL);
      if ($signingEnabled === NULL) {
        $signingEnabled = $idplocalConfig->getBoolean('sign.logout', NULL);
      }
    } elseif ($message instanceof SAML2\SAML2_EidasAuthnRequest) {
      $signingEnabled = $splocalConfig->getBoolean('sign.authnrequest', NULL);
      if ($signingEnabled === NULL) {
        $signingEnabled = $idplocalConfig->getBoolean('sign.authnrequest', NULL);
      }
    }
    if ($signingEnabled === NULL) {
      $signingEnabled = $idplocalConfig->getBoolean('redirect.sign', NULL);
      if ($signingEnabled === NULL) {
        $signingEnabled = $splocalConfig->getBoolean('redirect.sign', FALSE);
      }
    }
    if (!$signingEnabled) {
      return;
    }
```

```
    self::addSign($splocalConfig, $idplocalConfig, $message);
  }
```

- *addSign* – Método encargado de buscar las claves y certificados indicados en los parámetros del SP y de añadir los correspondientes al mensaje. Se basa en métodos de la clase *SimpleSAML_Utilities* para el parseo y manipulación del mensaje y en objetos de la clase *XMLSecurityKey* para la manipulación de las claves.

```
public static function addSign(SimpleSAML_Configuration $splocalConfig,
SimpleSAML_Configuration $idplocalConfig, SAML2\SignedElement $element) {
    $keyArray = SimpleSAML_Utilities::loadPrivateKey($splocalConfig, TRUE);
    $certArray = SimpleSAML_Utilities::loadPublicKey($splocalConfig, FALSE);
    $privateKey = new \XMLSecurityKey(\XMLSecurityKey::RSA_SHA512, array('type' =>
'private'));
    if (array_key_exists('password', $keyArray)) {
      $privateKey->passphrase = $keyArray['password'];
    }
    $privateKey->loadKey($keyArray['PEM'], FALSE);

    $element->setSignatureKey($privateKey);

    if ($certArray === NULL) {
      /* We don't have a certificate to add. */
      return;
    }
```

6.6.6. Construcción de la petición de Logout

Para realizar una solicitud de cierre de sesión, se debe generaruna petición SAML similar a la de autenticación pero, en este caso, de tipo logout.

En logout.php, se construye la petición partiendo de la configuración local e indicando el endpoint de la url que deberá recibir y procesar la respuesta, y la url del destino:

```
public static function buildLogoutRequest(SimpleSAML_Configuration $srclocalConfig,
SimpleSAML_Configuration $dstlocalConfig) {

              $lr = new SAML2\LogoutRequest();
              $lr->setIssuer($srclocalConfig->getString('logout.url'));
               $lr->setDestination($srclocalConfig->getString('idp'));
              self::addRedirectSignLogout($srclocalConfig, $dstlocalConfig, $lr);

              return $lr;
    }
```

6.6.7. Construcción del ticket SAML

A partir del objeto SAML2_EidasAuthnRequest obtenido previamente, se procede a la construcción del ticket SAML.

- **SAML2_EidasHTTPost** – Clase encargada de implementar el HTTP POST Binding. Para ello utilizará el método sendDefault o send, dependiendo si los atributos a solicitar han sido seleccionados por el usuario o si se han seleccionado por defecto.

```
$b = new SAML2_EidasHTTPPost($_POST['country'], $_POST);
$b->send($ar);
```

  - *Send* y *sendDefault* – Tienen como parámetro de entrada el objeto *SAML2_EidasAuthnRequest*. A partir de la información proporcionada en dicho objeto, convierten el mensaje en un XML inicialmente no firmado mediante el método *toUnsignedXML* y posteriormente firmado gracias al método *toSignedXML*. Ambos métodos pertenecen a la clase *SAML2_Message.*

```
public function toUnsignedXML()
    {
        $this->document = DOMDocumentFactory::create();
        $root = $this->document->createElementNS(Constants::NS_SAMLP, 'saml2p:'.$this-
>tagName);
        $this->document->appendChild($root);

        $root->setAttributeNS(Constants::NS_SAML, 'saml2:tmp', 'tmp');
        $root->removeAttributeNS(Constants::NS_SAML, 'tmp');

        $root->setAttributeNS(Constants::NS_EIDAS, 'eidas:tmp', 'tmp');
        $root->removeAttributeNS(Constants::NS_EIDAS, 'tmp');

        $root->setAttribute('ID', $this->id);
        $root->setAttribute('Version', '2.0');
        $root->setAttribute('IssueInstant', gmdate('Y-m-d\TH:i:s\Z', $this->issueInstant));

        if ($this->destination !== null) {
            $root->setAttribute('Destination', $this->destination);
        }

        $root->setAttribute('ForceAuthn', 'true');

        if ($this->consent !== null && $this->consent !== Constants::CONSENT_UNSPECIFIED) {
            $root->setAttribute('Consent', $this->consent);
        }

        if ($this->issuer !== null) {
            if (is_string($this->issuer)) {
                Utils::addString($root, Constants::NS_SAML, 'saml2:Issuer', $this->issuer);
            } elseif ($this->issuer instanceof XML\saml\Issuer) {
                $this->issuer->toXML($root);
            }
        }

        if (!empty($this->extensions)) {
            Extensions::addList($root, $this->extensions);
        }
        return $root;
    }
```

```
public function toSignedXML()
    {
```

```
         $root = $this->toUnsignedXML();

        if ($this->signatureKey === null) {
            /* We don't have a key to sign it with. */

            return $root;
        }

        /* Find the position we should insert the signature node at. */
        if ($this->issuer !== null) {
            /*
             * We have an issuer node. The signature node should come
             * after the issuer node.
             */
            $issuerNode = $root->firstChild;
            $insertBefore = $issuerNode->nextSibling;
        } else {
            /* No issuer node - the signature element should be the first element. */
            $insertBefore = $root->firstChild;
        }

        Utils::insertSignature($this->signatureKey, $this->certificates, $root,
$insertBefore);

        return $root;
    }
```

- *insertSignature* - Es el método utilizados para insertar el nodo Signature en el XML y pertenece a la clase *Utils*. Sus parámetros de entrada son; la clave utilizada para firmar el mensaje, los certificados que deben ser añadidos al nodo Signature, el nodo XML a firmar y el elemento XML que se tomara como referencia para colocar este nodo.

```
public static function insertSignature(
        \XMLSecurityKey $key,
        array $certificates,
        \DOMElement $root,
        \DOMNode $insertBefore = null
    ) {
        $objXMLSecDSig = new \XMLSecurityDSig();
        $objXMLSecDSig->setCanonicalMethod(\XMLSecurityDSig::EXC_C14N);

        switch ($key->type) {
            case \XMLSecurityKey::RSA_SHA256:
                $type = \XMLSecurityDSig::SHA256;
                break;
            case \XMLSecurityKey::RSA_SHA384:
                $type = \XMLSecurityDSig::SHA384;
                break;
            case \XMLSecurityKey::RSA_SHA512:
                $type = \XMLSecurityDSig::SHA512;
                break;
            default:
                $type = \XMLSecurityDSig::SHA1;
        }
        $objXMLSecDSig->addReferenceList(
            array($root),
            $type,
```

```
            array('http://www.w3.org/2000/09/xmldsig#enveloped-signature',
\XMLSecurityDSig::EXC_C14N),
            array('id_name' => 'ID', 'overwrite' => false)
        );

        $objXMLSecDSig->sign($key);

        foreach ($certificates as $certificate) {
            $objXMLSecDSig->add509Cert($certificate, true);
        }

        $objXMLSecDSig->insertSignature($root, $insertBefore);
    }
```

- ***XMLSecurityDSig, XMLSecurityKey y XMLSecEnc*** – Clases utilizadas para el trabajo con claves.

Una vez generado el ticket SAML se produce el envío de la petición:

```
    public function sendDefault(SAML2\SAML2_EidasAuthnRequest $message) {

              if ($this->destination === NULL) {
                     $destination = $message->getDestination();
              } else {
                     $destination = $this->destination;
              }

             if ($this->country !== NULL && $this->country !== "") {
               $destination = $destination /*. '?country=' . $this->country*/;
             }

              $relayState = $message->getRelayState();

                //This method sign the resulting XML document if the private key for the
signature is set.
              $msgStr = $message->toSignedXML();
              $msgStr = $msgStr->ownerDocument->saveXML($msgStr);

              SimpleSAML_Utilities::debugMessage($msgStr, 'out');

              $msgStr = base64_encode($msgStr);
              $msgStr = htmlspecialchars($msgStr);

              if ($message instanceof SAML2\SAML2_EidasAuthnRequest) {
                     $msgType = 'SAMLRequest';
              } else {
                     $msgType = 'SAMLResponse';
              }

              $destination = htmlspecialchars($destination);

              if ($relayState !== NULL) {
                     $relayState = '<input type="hidden" name="RelayState" value="' .
htmlspecialchars($relayState) . '">';
              } else {
                     $relayState = '';
              }
                /**
                 * The output is the final SAML Authentication Request
                 */
              $out = <<<END
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>POST data</title>
</head>
<body onload="document.forms[0].submit()">
<noscript>
<p><strong>Note:</strong> Since your browser does not support JavaScript, you must press the
button below once to proceed.</p>
</noscript>
<form method="post" action="$destination">
<input type="hidden" name="$msgType" value="$msgStr" />

<input type="hidden" name="country" value="ES"/>

$relayState
<noscript><input type="submit" value="Submit" /></noscript>
</form>
</body>
</html>
END;
              echo($out);
              exit(0);
       }
```

6.6.8. Validación de una respuesta

Para la validación de una SAML Response, se utilizan las siguientes clases:

- **HTTPPost** – Subclase que extiende de *Binding*.
  - *receive* – Recibe un mensaje SAML 2 enviado una petición HTTP-POST binding.

```
public function receive()
    {
        if (array_key_exists('SAMLRequest', $_POST)) {
            $msg = $_POST['SAMLRequest'];
        } elseif (array_key_exists('SAMLResponse', $_POST)) {
            $msg = $_POST['SAMLResponse'];
        } else {
            throw new \Exception('Missing SAMLRequest or SAMLResponse parameter.');
        }

        $msg = base64_decode($msg);

        Utils::getContainer()->debugMessage($msg, 'in');

        $document = DOMDocumentFactory::fromString($msg);
        $xml = $document->firstChild;

        $msg = SAML2_Message::fromXML($xml);

        if (array_key_exists('RelayState', $_POST)) {
            $msg->setRelayState($_POST['RelayState']);
        }

        return $msg;
```

```
    }
```

- **eidas_saml_Message:** Se vuelve a utilizar un método de esta clase anteriormente definida, checkSign
  - *checkSign* - Método encargado de la validación de la firma del mensaje recibido.

```
public static function checkSign(SimpleSAML_Configuration $splocalConfig,
SAML2\SignedElement $element) {

		/* Find the public key that should verify signatures by this entity. */
		$keys = $splocalConfig->getPublicKeys('signing');
		if ($keys !== NULL) {
			$pemKeys = array();
			foreach ($keys as $key) {
				switch ($key['type']) {
				case 'X509Certificate':
					$pemKeys[] = "-----BEGIN CERTIFICATE-----\n" .
						chunk_split($key['X509Certificate'], 64) .
						"-----END CERTIFICATE-----\n";
					break;
				default:
					SimpleSAML_Logger::debug('Skipping unknown key type: ' .
$key['type']);
				}
			}

		} elseif ($srcMetadata->hasValue('certFingerprint')) {
			$certFingerprint = $srcMetadata->getArrayizeString('certFingerprint');
			foreach ($certFingerprint as &$fp) {
				$fp = strtolower(str_replace(':', '', $fp));
			}

			$certificates = $element->getCertificates();

			/*
			 * We don't have the full certificate stored. Try to find it
			 * in the message or the assertion instead.
			 */
			if (count($certificates) === 0) {
				SimpleSAML_Logger::debug('No certificate in message when
validating against fingerprint.');
				return FALSE;
			} else {
				SimpleSAML_Logger::debug('Found ' . count($certificates) . '
certificates in ' . get_class($element));
			}

			$pemCert = self::findCertificate($certFingerprint, $certificates);
			$pemKeys = array($pemCert);
		} else {
			throw new SimpleSAML_Error_Exception(
				'Missing certificate in metadata for ' .
				var_export($splocalConfig->getString('entityid'), TRUE));
		}

		SimpleSAML_Logger::debug('Has ' . count($pemKeys) . ' candidate keys for
validation.');

		$lastException = NULL;
		foreach ($pemKeys as $i => $pem) {
```

```
                    $key = new XMLSecurityKey(XMLSecurityKey::RSA_SHA512,
array('type'=>'public'));
                    $key->loadKey($pem);

                    try {
                            /*
                             * Make sure that we have a valid signature on either the
response
                             * or the assertion.
                             */
                            $res = $element->validate($key);
                            if ($res) {
                                    SimpleSAML_Logger::debug('Validation with key #' . $i . '
succeeded.');
                                    return TRUE;
                            }
                            SimpleSAML_Logger::debug('Validation with key #' . $i . ' failed
without exception.');
                    } catch (Exception $e) {
                            SimpleSAML_Logger::debug('Validation with key #' . $i . ' failed
with exception: ' . $e->getMessage());
                            $lastException = $e;
                    }
            }
```

- **Crypto:** Clase que describe algunos de los métodos relacionados con la criptografía.
  - *loadPrivateKey* – Carga una clave privada de los metadatos.
- **XMLSecurityKey:** Clase perteneciente a la librería publica xmlseclibs anteriormente descrita y utilizada en el tratamiento de claves de seguridad y XML. En este caso se crea un nuevo objeto de esta clase para intervenir en la parte de descifrado del mensaje.
  - *RSA_OAEP_MGF1P* – Constante que indica el algoritmo a usar para desencriptar el mensaje.
  - *LoadKey* – Método que carga la clave que cumpla con los parámetros de entrada indicados.
- **EncryptedAssertion** – Clase encargada de manejar las respuestas encriptadas.
  - *getAssertion* – Método para extraer la respuesta.
- **Utils** – Clase previamente descrita.
  - *decryptElement* y *doDecryptElement*– Métodos utilizado para desencriptar el mensaje.

```
//Obtener las assertions en caso de qe sean un objeto EncryptedAsserton, es decir, descifrar
las assertions
        if ($assertions[0] instanceof SAML2\EncryptedAssertion) {
            try {

                                            $keyArray =
SimpleSAML\Utils\Crypto::loadPrivateKey($metadata, TRUE);
                                            assert('isset($keyArray["PEM"])');

                                            $key = new
XMLSecurityKey(XMLSecurityKey::RSA_OAEP_MGF1P, array('type'=>'private'));
                                            if (array_key_exists('password', $keyArray)) {
                                                    $key->passphrase =
$keyArray['password'];
                                            }
                                            $key->loadKey($keyArray['PEM']);
```

```
                                                $assertions[0] = $assertions[0]-
>getAssertion($key, array());


                                            }
```

Por último, el método que se encarga en última instancia de desencriptar el mensaje es *doDecryptElement* cuyos parámetros de entrada son, los datos encriptados, la clave necesaria y un parámetro opcional que permite introducir una lista negra de algoritmos de encriptación.

```
//Obtener las assertions en caso de qe sean un objeto EncryptedAsserton, es decir, descifrar
las assertions

private static function doDecryptElement(\DOMElement $encryptedData, \XMLSecurityKey
$inputKey, array &$blacklist)
    {
        $enc = new \XMLSecEnc();

        $enc->setNode($encryptedData);
        $enc->type = $encryptedData->getAttribute("Type");
       //  $encryptedNode = $encryptedData->firstChild;
        $symmetricKey = $enc->locateKey($encryptedData);

        if (!$symmetricKey) {
            throw new \Exception('Could not locate key algorithm in encrypted data.');
        }

        $symmetricKeyInfo = $enc->locateKeyInfo($symmetricKey);
        if (!$symmetricKeyInfo) {
            throw new \Exception('Could not locate <dsig:KeyInfo> for the encrypted key.');
        }

        $inputKeyAlgo = $inputKey->getAlgorith();
        if ($symmetricKeyInfo->isEncrypted) {
            $symKeyInfoAlgo = $symmetricKeyInfo->getAlgorith();

            if (in_array($symKeyInfoAlgo, $blacklist, true)) {
                throw new \Exception('Algorithm disabled: ' . var_export($symKeyInfoAlgo,
true));
            }

            if ($symKeyInfoAlgo === \XMLSecurityKey::RSA_OAEP_MGF1P && $inputKeyAlgo ===
\XMLSecurityKey::RSA_1_5) {
                /*
                 * The RSA key formats are equal, so loading an RSA_1_5 key
                 * into an RSA_OAEP_MGF1P key can be done without problems.
                 * We therefore pretend that the input key is an
                 * RSA_OAEP_MGF1P key.
                 */
                $inputKeyAlgo = \XMLSecurityKey::RSA_OAEP_MGF1P;
            }

            /* Make sure that the input key format is the same as the one used to encrypt
the key. */
            if ($inputKeyAlgo !== $symKeyInfoAlgo) {
                throw new \Exception(
                    'Algorithm mismatch between input key and key used to encrypt ' .
                    ' the symmetric key for the message. Key was: ' .
                    var_export($inputKeyAlgo, true) . '; message was: ' .
                    var_export($symKeyInfoAlgo, true)
```

```
                );
            }

            /** @var XMLSecEnc $encKey */
            $encKey = $symmetricKeyInfo->encryptedCtx;
            $symmetricKeyInfo->key = $inputKey->key;

            $keySize = $symmetricKey->getSymmetricKeySize();
            if ($keySize === null) {
                /* To protect against "key oracle" attacks, we need to be able to create a
symmetric key, and for that we need to know the key size.
                 */
                throw new \Exception('Unknown key size for encryption algorithm: ' .
var_export($symmetricKey->type, true));
            }

            try {
                $key = $encKey->decryptKey($symmetricKeyInfo);
                if (strlen($key) != $keySize) {
                    throw new \Exception(
                        'Unexpected key size (' . strlen($key) * 8 . 'bits) for encryption
algorithm: ' .
                        var_export($symmetricKey->type, true)
                    );
                }
            } catch (\Exception $e) {
                /* We failed to decrypt this key. Log it, and substitute a "random" key. */
                Utils::getContainer()->getLogger()->error('Failed to decrypt symmetric key:
' . $e->getMessage());
                /* Create a replacement key, so that it looks like we fail in the same way
as if the key was correctly padded. */

                /* We base the symmetric key on the encrypted key and private key, so that
we always behave the same way for a given input key.
                 */
                $encryptedKey = $encKey->getCipherValue();
                $pkey = openssl_pkey_get_details($symmetricKeyInfo->key);
                $pkey = sha1(serialize($pkey), true);
                $key = sha1($encryptedKey . $pkey, true);

                /* Make sure that the key has the correct length. */
                if (strlen($key) > $keySize) {
                    $key = substr($key, 0, $keySize);
                } elseif (strlen($key) < $keySize) {
                    $key = str_pad($key, $keySize);
                }
            }
            $symmetricKey->key=$key;
        } else {
            $symKeyAlgo = $symmetricKey->getAlgorith();
            /* Make sure that the input key has the correct format. */
            if ($inputKeyAlgo !== $symKeyAlgo) {
                throw new \Exception(
                    'Algorithm mismatch between input key and key in message. ' .
                    'Key was: ' . var_export($inputKeyAlgo, true) . '; message was: ' .
                    var_export($symKeyAlgo, true)
                );
            }
            $symmetricKey = $inputKey;
        }

        $algorithm = $symmetricKey->getAlgorith();
```

```
        if (in_array($algorithm, $blacklist, true)) {
            throw new \Exception('Algorithm disabled: ' . var_export($algorithm, true));
        }

        /** @var string $decrypted */
        $decrypted = $enc->decryptNode($symmetricKey, false);

        /*
         * This is a workaround for the case where only a subset of the XML
         * tree was serialized for encryption. In that case, we may miss the
         * namespaces needed to parse the XML.
         */
        $xml = '<root xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" '.
                     'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' .
            $decrypted .
            '</root>';

        try {
            $newDoc = DOMDocumentFactory::fromString($xml);
        } catch (RuntimeException $e) {
            throw new \Exception('Failed to parse decrypted XML. Maybe the wrong sharedkey
was used?', 0, $e);
        }

        $decryptedElement = $newDoc->firstChild->firstChild;
        if ($decryptedElement === null) {
            throw new \Exception('Missing encrypted element.');
        }

        if (!($decryptedElement instanceof \DOMElement)) {
            throw new \Exception('Decrypted element was not actually a \DOMElement.');
        }

        return $decryptedElement;
```

6.6.9. Validación de una respuesta de logout

Una vez realizada una petición de logout, el endpoint recibirá una respuesta y en **logoutResponse.php** se procesará la misma.

Como se ve a continuación, se validará la firma de manera similar a una SAML Reponse y se devolverá error en caso de que la respuesta sea inválida, o se redirigirá al usuario a la página principal para que pueda iniciar sesión de nuevo:

```
          try {
        $retVal = eidas_saml_Message::checkSign($localConfig, $response);

        if (!$response->isSuccess() || !$retVal) {
                $statsData['error'] = $response->getStatus();
                throw Exception("LogoutResponse is Fail or invalid");
        }
        $relayState = $response->getRelayState();
        header ('location: /SP');
    } catch (Exception $e) { // TODO: look for a specific exception
            echo "<h2>Se ha producido un error</h2><br>Por favor, haga clic <a
href=\"/SP/\">aqu&iacute;</a> para volver a la p&aacute;gina principal.";
            throw $e; // do not ignore other exceptions!
    }
```

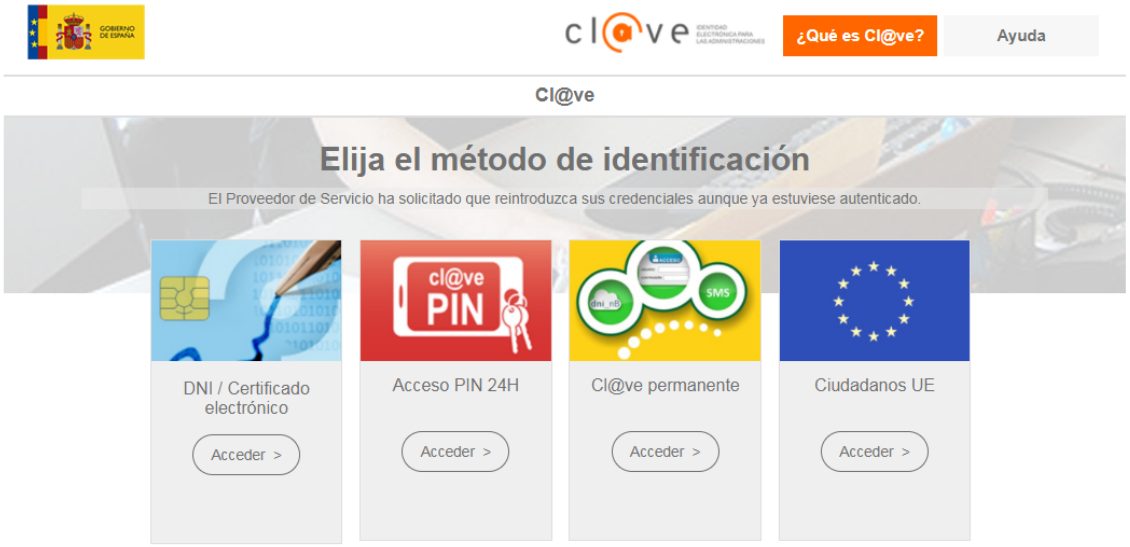
7. Pasos durante la ejecución de una prueba

Al acceder al SP de demo desde el navegador se le muestra al usuario una ventana con un boton: “Iniciar sesión”.



En caso de que se pulse sobre la opción “Iniciar sesión”, el programa iniciará automáticamente el proceso de enviar una petición SAML por defecto sin que sea necesario intervenir. Esta petición solicitará los atributos requeridos como atributos obligatorios y el resto como opcionales, excluyendo los atributos de representación.

Una vez que el usuario selecciona los atributos deseados, habrá que presionar el botón “Iniciar sesión”, que llevará al usuario hacia la pasarela Clave.



Tras realizar el proceso de identificación, y tras recibir la respuesta, se muestra una página en la que, tras validar el ticket recibido, se muestra la página final del ciclo, donde se debería continuar con el flujo de negocio normal de una aplicación. En el ejemplo los valores recabados.

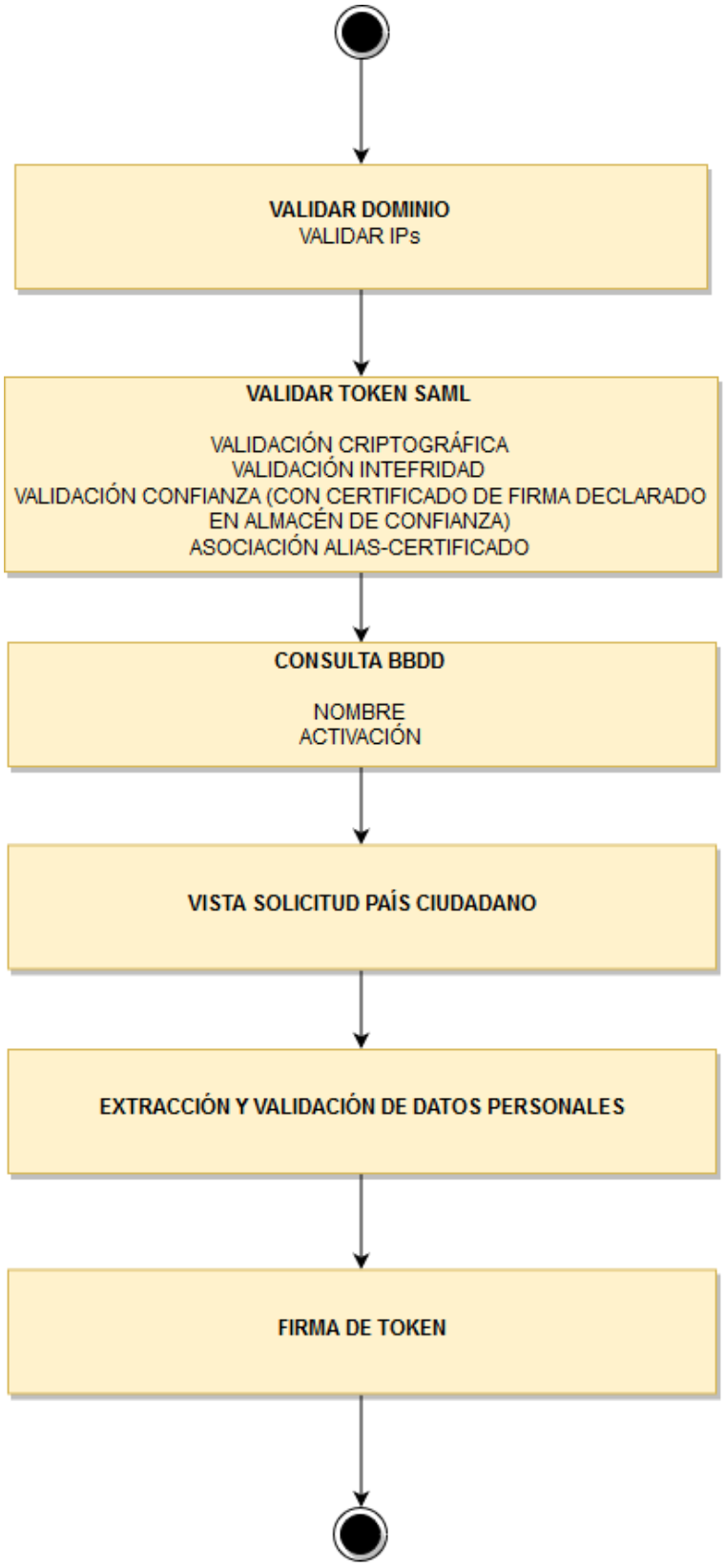


En caso de error se mostrará una página descriptiva del problema recibido.



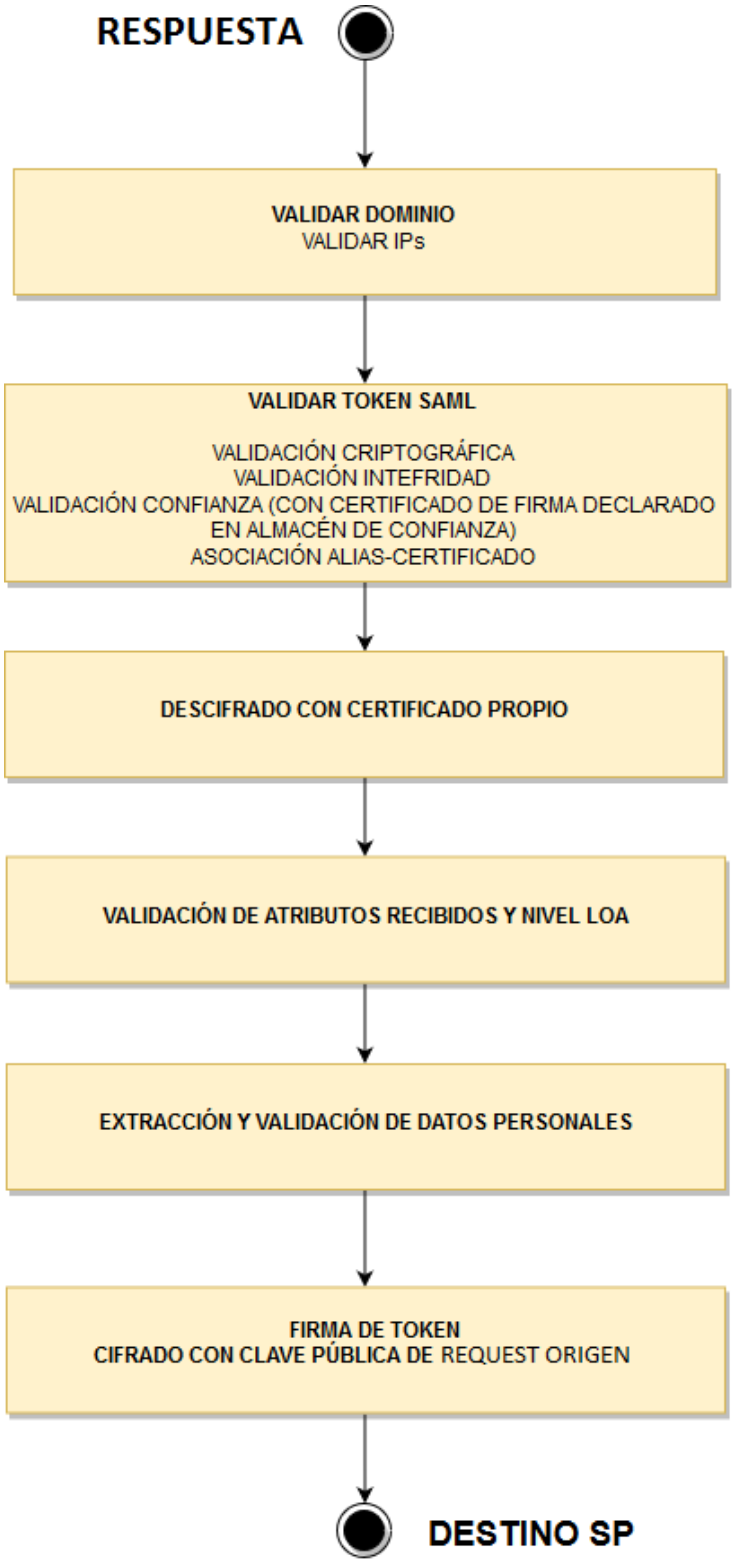
7.1. Validar Request

En caso de haber recibido una solicitud, se sigue el siguiente flujo lógico.



7.2. Validar respuesta

En caso de haber recibido una respuesta SAML, se procesa del modo descrito a continuación:



8. Información de conexión a la pasarela Cl@ve

Servicios Estables (entorno de pruebas conectado a Internet):

- URL de servicio de la plataforma Cl@ve:
    - https://se-pasarela.clave.gob.es/Proxy2/ServiceProvider

Producción

- URL de servicio de la pasarela Cl@ve
    - https://pasarela.clave.gob.es/Proxy2/ServiceProvider

9. Anexo

En este apartado se va a hablar de cómo se firman y cifran XML con el protocolo SAML2 y metadatos con el mismo protocolo.

Para la firma de un XML, se utiliza la clave privada del mismo que los crea, lo que incluye un elemento en el XML llamado *Signature* que contiene una serie de elementos como es el *DigestMethod*, que contiene un atributo *Algorithm* el cual contiene el método que se ha utilizado para firmar (por ejemplo, el atributo podría ser: *Algorithm =*[*http://www.w3.org/2001/04/xmlenc#sha512*](http://www.w3.org/2001/04/xmlenc#sha512), lo cual nos dice que se ha firmado con sha512).

Si se diese el caso de que alguien modificase el XML después de realizar la firma, ésta ya no sería válida, ya que los datos sobre los que se firmó son diferentes a los que hay una vez modificado el XML.

```
− <md:EntityDescriptor entityID="https://se-eidas2.redsara.es/EidasNode/ConnectorResponderMetadata" validUntil="2017-09-29T07:52:42.214Z">
  − <ds:Signature>
    − <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha512"/>
      − <ds:Reference URI="">
        + <ds:Transforms></ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
        − <ds:DigestValue>
            DGu9FLTmvkrHFUXrYV3OAD2hfFFa8UFB3tdbiHlxNGItSzyiK/SljuhMSb9mnAKnWcK/M+zynUPR8qEbi6pbxg==
          </ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
    + <ds:SignatureValue></ds:SignatureValue>
    + <ds:KeyInfo></ds:KeyInfo>
    </ds:Signature>
  + <md:Extensions></md:Extensions>
  + <md:IDPSSODescriptor WantAuthnRequestsSigned="true" protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"></md:IDPSSODescriptor>
  + <md:Organization></md:Organization>
  + <md:ContactPerson contactType="support"></md:ContactPerson>
  + <md:ContactPerson contactType="technical"></md:ContactPerson>
  </md:EntityDescriptor>
```

El proceso para comprobar la firma será el siguiente:

- Ver el algoritmo que se utiliza, y si usa el mismo un extremo que el otro se continúa.
- Se recoge el nodo X509Certificate, que es la clave pública del extremo firmante. Si no está incluida en el almacén de claves de CAs de confianza, no se continúa.
- Con ella, se comprueba la firma de la parte del XML firmada.
- El siguiente paso es comprobar que soporta todos los atributos obligatorios para personas físicas y jurídicas.

Una vez llegados a este punto con todas las validaciones, se puede dar por terminado el método de comprobar una firma.

El proceso para cifrar un XML, se puede elegir cifrar sólo una parte del mismo o todo él dejando en el atributo *URI* del nodo *Reference*. Si se deja vacío, se cifrará todo el XML.

Se coge la clave pública del otro extremo y se cifra el XML.

Una vez cifrado, no se pueden realizar cambios sobre el mismo, ya que el cifrado no valdría.

El proceso para firmar un XML es el siguiente:

- Se crea el XML con el protocolo SAML2.
- Se cifra el XML (completo, o sólo una parte del mismo) con la clave pública del otro extremo.
- Se firma ese XML firmado con la propia clave privada.
- Una vez hecho esto, se puede enviar al otro extremo.
- El proceso si el XML está firmado y cifrado, primero se comprobará la firma con los pasos ya explicados y después se descifrará. Es decir, el proceso completamente contrario. Si primero si cifra y después se firma, aquí se comprueba la firma y después se descifra el contenido para obtener el XML en claro.

9.1. Algoritmos criptográficos empleados

En este apartado se va a hablar de cómo se firman y cifran XML con el protocolo SAML2 y metadatos, explicando los algoritmos elegidos y cómo funcionan.

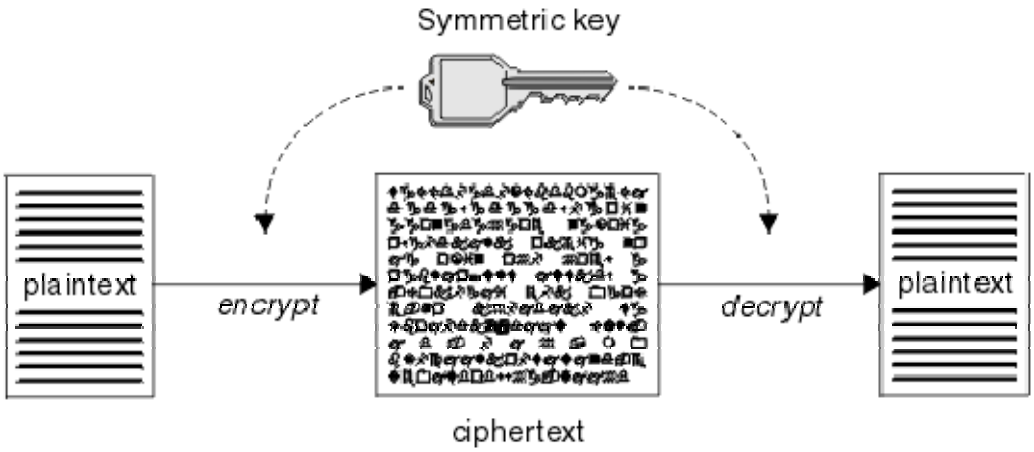
9.1.1. Cálculo de *digest*

Un algoritmo de digest consiste en una función matemática que obtiene un resumen de los datos procesados. El resultado se llama “digest‟, “hash” o huella digital. El valor calculado debe ser único, es decir, no debe existir una entrada diferente que dé como resultado el mismo digest. Se utiliza para obtener un identificador único de tamaño fijo de un recurso y para evidenciar si se ha producido un cambio en los datos.

- Se emplean estándares definidos por NIST.
- Se asocia a la integridad de los datos.
- Algoritmos y familias frecuentes:
    - MD-5, SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) y el nuevo SHA-3.
    - Existen ataques conocidos para MD-5 y SHA-1 que hacen que su uso no esté recomendado.
- Cl@ve utiliza el algoritmo SHA-512.

9.1.2. Cifrado simétrico

En el cifrado simétrico se aplican una serie de transformaciones a los datos a proteger. Estas transformaciones consisten en permutaciones y transposiciones realizadas secuencialmente en función de una condición previa, es decir, de una clave, de tal forma que el proceso sea irreversible sin conocer la clave utilizada. Para cifrar y descifrar la información se utiliza la misma clave.



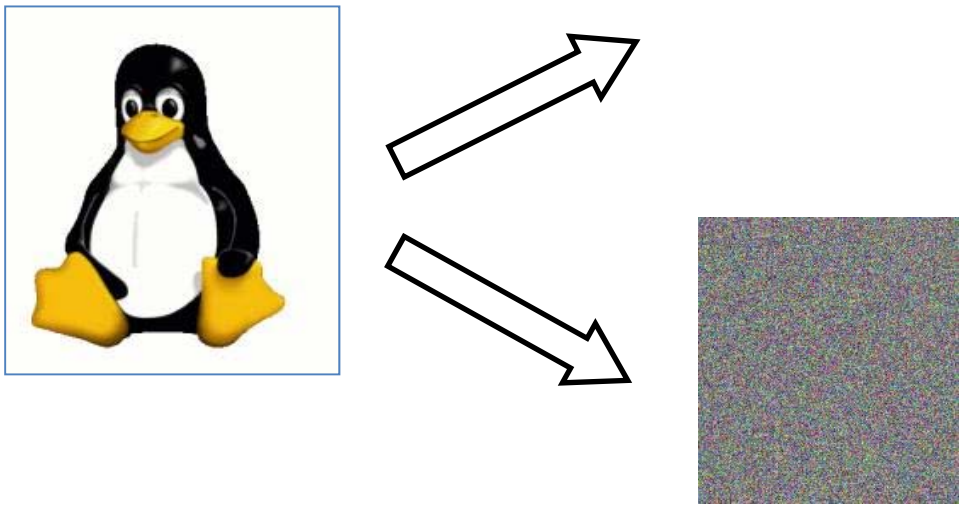
9.1.3. Cifrado simétrico de tipo AES (Rijndael)

Se trata del estándar de cifrado adoptado por el protocolo eIDAS. Soporta longitudes de clave de 128, 192 y 256 bits. Es el sustituto de Data Encryption Standard (DES). Y está ampliamente estudiado y analizado por la comunidad internacional. Uno de sus puntos fuertes para su elección fue el rendimiento. Su definición se encuentra en el FIPS-197 y actualmente se considera el algoritmo de criptografía simétrica más extendido:

- Principales características:
    - Procesa la información en bloques de 128 bits.
    - Basado en permutaciones y substituciones utilizando un esquema matricial de 4x4.
    - La clave define las operaciones de permutación y sustitución que se realizan durante el cifrado.
    - La longitud de clave está ligada al número de operaciones intermedias que se realizan
        - AES128 → 10 ciclos de repetición
        - AES196 → 12 ciclos de repetición
        - AES256 → 14 ciclos de repetición

Es remarcable llegados a este punto que, en contra de lo que podría suponerse, aplicar un cifrado simétrico fuerte no es suficiente, ya que la información cifrada puede contener patrones que un atacante podría explotar para romper el cifrado. Este punto se puede apreciar en la siguiente imagen, que compara un cifrado crudo y un cifrado bien inicializado, alimentado de ruido, etc.



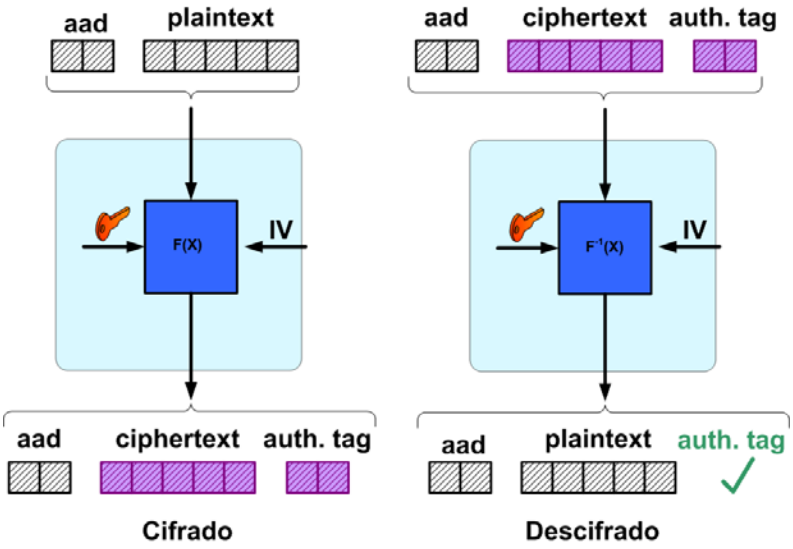


Para inicializar correctamente el cifrado se introducen los siguientes conceptos asociados:

- Vector de inicialización (IV): estado inicial del cifrador. Es un valor aleatorio.
- Existen requisitos sobre los IVs en función del modo de cifrado.
  - Los IVs deben ser conocidos por ambas partes para poder descifrar la información
  - Los IVs no necesariamente tienen que ser secretos. En muchas ocasiones forman parte de las cabeceras de los protocolos.
- Padding: relleno para completar el tamaño de bloque. Hay que ser cuidadoso al aplicar el padding porque puede introducir ciertos patrones en los datos.
- Nonce: valor único para cada ejecución.

9.1.4. Modo de cifrado GCM (Galois Counter Mode).

Este es el módulo de cifrado utilizado por Cl@ve para cifrar aserciones. En este modo de cifrado, además de cifrar con AES y RSA, se calcula una etiqueta de autenticación de los datos que permite validar la autenticidad.



- Características:
  - Procesado por paquetes.
  - Soporta distintas longitudes para la etiqueta de autenticación.
  - Necesario un IV por paquete.
  - Longitud recomendada IV: 96 bits
  - IVs no pueden repetirse para una misma clave.

9.1.5. Cifrado asimétrico

El cifrado asimétrico tiene un objetivo similar al cifrado simétrico, pero divide la clave utilizada en dos partes. Lo que se cifra con una parte, solo puede ser descifrado con la otra parte, y viceversa. Su implementación se basa en problemas de la Teoría de Números y su fortaleza reside en la dificultad a la hora de factorizar grandes números enteros. Dicho de otro modo:

- Dados dos números primos grandes M y N es fácil hallar su producto P=MxN.
- Pero dado P es muy difícil hallar sus factores M y N. Se consideran problemas computacionalmente inabordables incluso con gran cantidad de recursos de potencia de cálculo y memoria.
- Actualmente un número de 1024 bits (≈310 dígitos) no puede ser factorizado:
    - Se estima que se necesitaría una máquina específicamente diseñada, con un coste >10M $, que tardaría un año en realizar la factorización de un solo número.

Esta clave en dos partes se utiliza para definir dos tipos de clave:

- Clave privada: conocida y custodiada únicamente por la entidad propietaria.
- Clave pública: publicada por la entidad propietaria y que cualquiera puede utilizar.

La algoritmia del cifrado asimétrico es mucho más costosa computacionalmente que la criptografía simétrica, por lo que no se suele utilizar para cifrar grandes cantidades de información.

Generalmente, las claves se asocian con el uso de certificados electrónicos. Los certificados electrónicos consisten en una serie de datos personales que, junto a la clave pública, son firmadas electrónicamente por una autoridad de certificación, dando lugar al ecosistema PKI (Pubic Key Infrastructure).

Una PKI permite formar cadenas de autenticación mediante cadenas de firmas, con las cuales, partiendo de la premisa de que se confía en la autoridad de certificación, es posible trasladar dicha confianza hasta la clave privada utilizada en un mensaje firmado o cifrado. El certificado electrónico asociado a dicha clave privada permite además asegurar que la persona o elemento referenciado es quien dice ser.

9.1.6. Cifrado RSA

Es un algoritmo de cifrado asimétrico desarrollado en la década de los 70 generalmente utilizado para cifrar mensajes pequeños y para realizar firma digital.

Para realizar un cifrado RSA, el emisor utiliza la clave pública del destinatario, que para descifrarlo, utiliza su clave privada. Nadie más puede leer el mensaje porque la clave privada solo la tiene el destinatario, mientras que cualquiera puede cifrar un mensaje para él. Dicho de otro modo, para realizar un cifrado asimétrico

- Una entidad A posee un par clave privada/clave pública (K / p)
- Si una entidad B desea enviar confidencialmente un mensaje M a la entidad A:
    - B conoce la clave pública de A p
    - B cifra el mensaje M con la clave p:
        - Z = CIFRARp(M)
    - B transmite Z a A.
- Sólo A puede acceder al contenido en claro del mansaje, ya que sólo A conoce su propia clave privada K
- A descifra con la clave K el mensaje Z recibido:
    - M = DESCIFRARK(Z).

Para realizar el descifrado asimétrico:

- Una entidad A posee un par clave privada/clave pública (K / p)
- Si A desea enviar un mensaje M, puede generar una firma F, de manera que cualquier otra entidad B pueda verificar que M fue emitido por A
    - A conoce su clave pública K
    - A genera la firma del mensaje M con la clave K:
    - F = FIRMAK(M)
    - A transmite el mensaje y la firma (M, F) a B
    - B recibe el mensaje M y verifica la firma de A con la clave p:
    - VERIFICARp(M, F)
    - Sólo A puede haber generado la firma F del mansaje M, ya que sólo A conoce su propia clave privada K
    - Por eficiencia en realidad no se firma el mensaje, sino el digest obtenido de aplicarle una función HASH.

Para realizar una firma electrónica, el emisor de la firma utiliza su clave privada para cifrar los datos a firmar. Compone un mensaje que incluye el dato original, su clave pública y el valor de la firma. El validador utiliza la clave pública para descifrar el valor de firma y compararlo con el hash de los datos originales. Si coinciden, tiene garantías sobre la procedencia del mensaje firmado. De esta manera se comprueba lo siguiente:

- Integridad: Tiene por objetivo detectar alteraciones, ya sean o no intencionadas.
- Autenticidad: Tiene por objetivo verificar la identidad/autoría.

- No repudio: Tiene por objetivo evitar la denegación de la autoría de la información o de una acción.

9.1.7. SSL/TLS

Es en primera instancia, la primera capa de seguridad que se aplica en el protocolo eIDAS, y que protege el canal utilizado para transmitir el token SAML y los metadatos. La protección consiste en el cifrado que aporta el uso del esquema "https" en detrimento de "http", dentro del navegador del usuario. Los sistemas de navegación web soportan una serie de algoritmos que les permita negociar con servidores. Estos algoritmos se agrupan en protocolos, entre los que se encuentran:

- SSL
    - Desarrollado por Netscape
    - Última versión 3.0 (1996)
- TLS:
    - Basado en SSL 3.0
    - Estándar del IETF (Internet Engineering Task Force)
    - Última versión 1.2 (RFC 5246)

El objetivo de estos protocolos consiste en proporcionar seguridad en las comunicaciones a través de una red. Se ejecuta en una capa entre el protocolo de aplicación (HTTP, SMTP, IMAP, etc.) y el protocolo de transporte. Combina varios algoritmos criptográficos:

- Criptografía asimétrica para establecimiento de claves simétricas.
- Criptografía simétrica para confidencialidad de los datos.
- Algoritmos MAC para la integridad de los datos.

Para establecer un canal de comunicaciones, un cliente y un servidor deciden establecer una conexión de tipo TLS, ya que ambos lo soportan. Comienza el handshake para negociar una clave simétrica compartida:

- El cliente manda un mensaje con sus capacidades criptográficas.
- El servidor manda un mensaje con los parámetros criptográficos, que se eligen basándose en las capacidades del cliente, junto con su certificado.
- Si el servidor requiere autenticación del cliente le solicita su certificado y verifica que es confiable para él.
- El cliente verifica que el certificado del servidor es confiable para él.
- Cliente y servidor negocian una clave simétrica (master) utilizando, por ejemplo, el algoritmo DH (Diffie-Hellman) con firma (ahora sobre un canal autenticado).
- El cliente y el servidor derivan una clave de sesión a partir de la clave master.
- El cliente y el servidor utilizan la clave de sesión para cifrar y autenticar los mensajes durante el resto de la conexión.

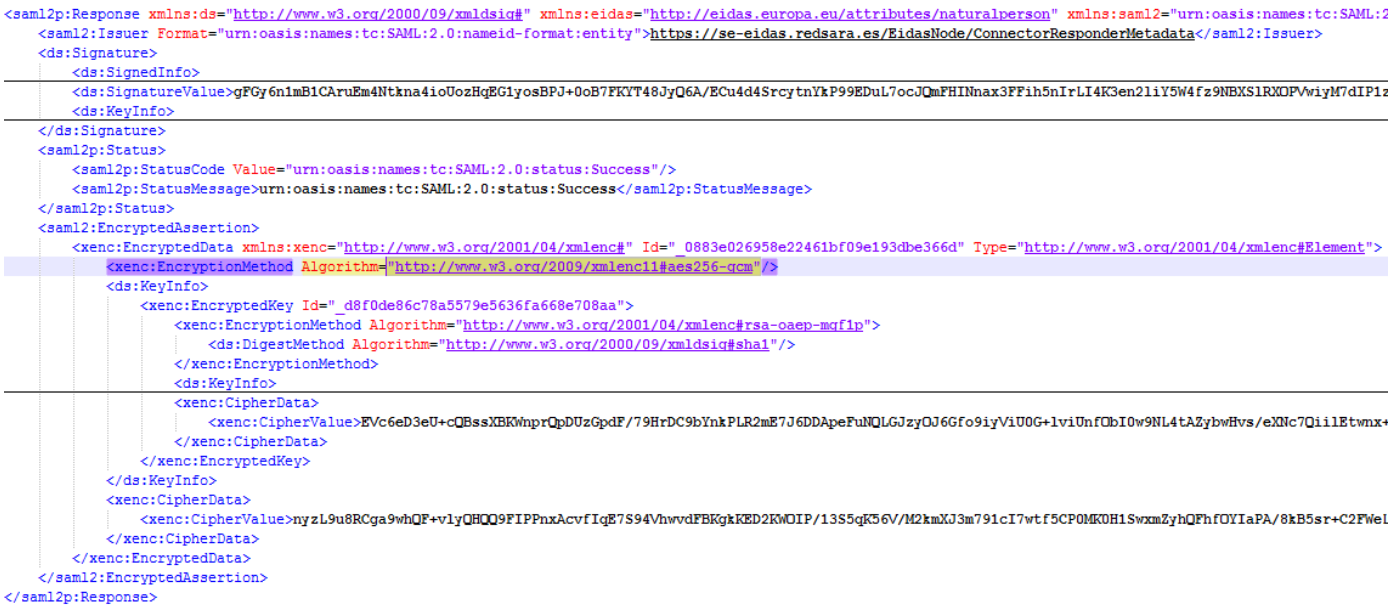
9.1.8. Firma y cifrado en el token SAML2

Para la firma de un XML, se incluye un elemento en el XML a firmar llamado *Signature.* Este elemento define su propio espacio de nombres y contiene una serie de sub-elementos que componen la firma. Entre ellos, están:

- El elemento *SignatureValue*, donde reside el valor de la firma.
- El elemento *SignedInfo*, donde residen las referencias hacia los datos firmados.
- El elemento el *SignatureMethod* y *DigestMethod*, que contiene un atributo *Algorithm* el cual contiene el método que se ha utilizado para firmar y calcular digest.
- El elemento *CanonicalizationMethod*, que especifica la manera de leer los datos.
- El elemento *KeyInfo*, que especifica la clave utilizada para firmar.

El proceso para firmar un XML es el siguiente:

- Se crea el XML según el estándar SAML2, perfil eIDAS, donde se indican:
  - Petición:
    - Una lista de atributos a recabar referidos al usuario.
    - Configuración de la autenticación (IdPs disponibles, nivel LoA mínimo, etc.).
    - Un identificador único del token SAML.
  - Respuesta:
    - Una lista de atributos obtenidos tras autenticar al usuario.
    - Datos propios de la autenticación. (IdP utilizado, nivel LoA logrado, etc.).
    - Opcionalmente, se cifra la aserción del token SAML, dentro del XML, con la clave pública del otro extremo, obtenida mediante consulta previa, y validación, de sus metadatos.
      - Se emplea el estándar https://www.w3.org/TR/xmlenc-core1/
      - Se cifran los datos con una clave simétrica precalculada y se adjuntan al token. En Cl@ve se utiliza el algortimo:
        "http://www.w3.org/2009/xmlenc11#aes256-gcm"
      - La clave anterior se cifra simétricamente usando la clave pública del destinatario y se adjunta al token. En Cl@ve se utiliza:
        “http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p”
      - Se incluye una referencia al destinatario de los datos cifrados.
    - Un atributo XML donde se hace referencia al ID único de la petición a la que se responde.
  - Ambos casos:
    - Datos identificativos del emisor (URL de metadatos, ProviderName, etc.).
    - Fecha de emisión, datos identificativos del receptor, un identificador único del token SAML, entre otros.
- Se firma ese XML. Se utiliza la clave privada propia del emisor, de forma que cualquiera pueda comprobar, mediante la clave pública, la autenticidad. Para realizar la firma se utiliza el estándar XMLDSig para generar una firma de tipo enveloped, donde la firma está dentro de lo firmado. Se adjunta el certificado X509 asociado a la clave utilizada, codificado en Base64, para facilitar el proceso de validación y reconstrucción de la confianza.
- Una vez hecho esto, se procede con el envío al otro extremo. Opcionalmente, como paso previo, se puede presentar al usuario la información a transmitir solicitando su consentimiento.
- A continuación, se muestra un ejemplo de XML firmado:



El proceso para comprobar la firma es el siguiente:

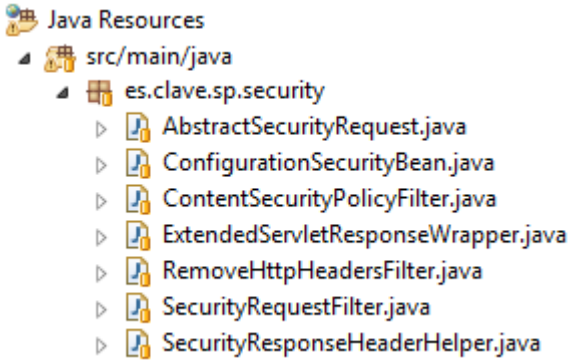
- El receptor abre el token recibido, valida los metadatos del emisor y procede a la validación de la firma electrónica.
- Se carga una instancia del algoritmo declarado en la propia firma.
- Se recoge el nodo XML llamado “X509Certificate”, donde reside la clave pública y el certificado del emisor. Este certificado, o su emisor, debe estar incluido en el almacén de confianza.
- Se recogen las referencias a los datos firmados y se calculan los valores de hash que les corresponden. Se comprueba la integridad comparando el valor indicado en la firma con el valor calculado durante la validación.
- Se utiliza la clave pública para descifrar el nodo XML llamado “SignatureValue”. Se calcula el digest del nodo “SignedInfo”. Se comprueba la autenticidad del mensaje comparando el valor digest descifrado con el valor calculado durante la validación.
- Si todo va bien, y en caso de que la respuesta venga cifrada, procede a descifrar el contenido para obtener la aserción XML en claro, donde se encuentran los datos personales del usuario. Para ello:
    - Se carga una instancia del algoritmo indicado en la aserción cifrada.
    - Se descifra asimétricamente la clave simétrica indicada utilizando la clave privada del receptor.
    - Se descifran los datos cifrados utilizando la clave simétrica obtenida del paso anterior.
- La validación del token incluye la validación de los atributos y del esquema eIDAS. Se comprueba que los datos obligatorios están presentes, etc. En la siguiente imagen se muestran algunos nodos que contienen esa información sobre los atributos, publicadas como metadatos de un módulo:

```
<saml2:Attribute FriendlyName="D-2012-17-EUIdentifier" Name="http://eidas.europa.eu/attributes/legalperson/D-2012-17-EUIdentifier"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="EORI" Name="http://eidas.europa.eu/attributes/legalperson/EORI" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="LEI" Name="http://eidas.europa.eu/attributes/legalperson/LEI" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="LegalAddress" Name="http://eidas.europa.eu/attributes/legalperson/LegalAddress"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="LegalName" Name="http://eidas.europa.eu/attributes/legalperson/LegalName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri"/>
<saml2:Attribute FriendlyName="LegalPersonIdentifier" Name="http://eidas.europa.eu/attributes/legalperson/LegalPersonIdentifier"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="SEED" Name="http://eidas.europa.eu/attributes/legalperson/SEED" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="SIC" Name="http://eidas.europa.eu/attributes/legalperson/SIC" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="TaxReference" Name="http://eidas.europa.eu/attributes/legalperson/TaxReference"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="VATRegistration" Name="http://eidas.europa.eu/attributes/legalperson/VATRegistration"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"/>
<saml2:Attribute FriendlyName="RepresentativeD-2012-17-EUIdentifier" Name="http://eidas.europa.eu/attributes/legalperson/representative/D-2012-17-EUIdentifier"
```

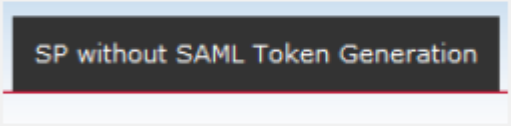
- Finalmente, el receptor ya dispone de los datos del usuario y continúa con el flujo de negocio que corresponda según sus propias necesidades.

9.2. Filtros WEB y configuración de seguridad

La gestión de seguridad web de los servlets se implementa en el siguiente paquete Java:



Contiene la lógica que permite filtrar el tráfico frente a solicitudes abusivas o maliciosas. Sus elementos de configuración aparecen en la siguiente página:

Key	Description
`security.header.CSP.enabled`	Enable/disable sending the Content Security Policy (CSP) header. CSP protects against the injection of foreign content (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.CSP.includeMozillaDirectives`	In the CSP, this additional directive can be added for backward compatibility with old Mozilla browsers (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.XXssProtection.block`	This header enables the cross-site-scripting (XSS) filter built into most recent web browsers (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.XContentTypeOptions.noSniff`	The only defined value 'nosniff' prevents Internet Explorer and Google Chrome from 'MIME-sniffing' by inspecting the content of a response (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.XFrameOptions.sameOrigin`	Prevents the application from being propagated in a frame or iframe, which in turns protects against key logging, clickjacking and similar attacks. Setting this option to **true** will prevent the eIDAS-Node from being framed in another application. If the SP needs to frame the eIDAS-Node, the option has to be set to 'false' (such as on the second tab of the SP Demo where the SAML request is generated by the eIDAS-Node).  (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.HSTS.includeSubDomains`	HTTP Strict-Transport-Security (HSTS) instructs browsers to prefer secure connections to the server (HTTP over SSL/TLS) over insecure ones (refer to the *eIDAS-Node Security Considerations* guide for more information about the security features).
`security.header.CSP.fallbackCheckMode`	If enabled, CSP fallback check mode includes an enforced CSP violation in JSP pages in order to check browser CSP feature. The included script displays a warning message in client browsers if CSP is not supported. However with CSP enabled browsers it may result in flood of warning messages logged by CSP report servlet. Disabled by default.

9.2.1. Gestión de ataques de denegación de servicio

En el fichero de configuración interno eidas-commons/src/main/resources/eidasParameters.properties se utiliza para activar o desactivar la validación de los siguientes parámetros de configuración.

```
max.spUrl.size=150
max.attrList.size=20000
max.invalidAttributeList.size=20000
max.attrName.size=100
max.callback.size=300
max.idp.url.size=300
max.attrValue.size=65535
max.attrType.size=25
max.spId.size=40
max.providerName.size=128
max.spName.size=25
max.country.size=150
max.qaaLevel.size=1
max.spQaaLevel.size=1
max.errorCode.size=5
max.errorMessage.size=300
max.username.size=30
max.connectorRedirectUrl.size=300
max.connector.redirectUrl.size=300
max.service.redirectUrl.size=300
max.serviceRedirectUrl.size=300
max.connectorAssertionUrl.size=300
max.SAMLRequest.size=131072
max.SAMLResponse.size=131072
max.RelayState.size=80
max.remoteAddr.size=300
max.remoteHost.size=300
max.localAddr.size=300
max.localName.size=300
```

Esta configuración limita en tamaño los datos que se intercambian y por lo tanto se definen en este documento, ya que pueden ocasionar posibles problemas de denegaciones de servicio ante una incorrecta configuración.

Se pueden destacar especialmente los campos max.SAMLResponse.size si se van a incorporar datos personales de considerable tamaño (este valor es heredado de Stork donde existe el antecedente de haber superado el tamaño descrito al incluir el suplemento al diploma académico, generando una incidencia general).

Este problema puede ser repetible con las URL ajenas cuando se incluyan parámetros en la URL o al hacer HTTP Redirect.

9.2.2. Ataques CSRF

En un ataque de CSRF, un eventual atacante aprovecha una sesión de autenticación previa que ha conseguido interceptar para impersonar a un usuario. Para preservar y transmitir un mensaje de solicitud como paso inicial de un protocolo SAML, se imponen ciertos requisitos para transmitir la respuesta. Concretamente**,** si un mensaje de solicitud de SAML va acompañado de un dato adicional "RelayState", el módulo debe devolver con su respuesta el mismo RelayState que recibió con la solicitud. Este campo ni se procesa ni se valida, solo se responde en modo eco.

Por ello, el formulario de solicitud incluye este campo junto a la solicitud SAML en Base64.

Al recibir la respuesta del IdP, se comprueba la integridad de su valor comparándolo con el valor original. Lo mismo debe hacer el SP.

9.3. Ataques en el XML

SQL Injection.- Se comprueba que no aparezca el carácter "'" en el XML.

Cross Site Scripting.- Se comprueba que no aparezca la palabra "script" en el XML.

Ataque DTD.- Existe un repositorio interno, nunca se descargan esquemas externos.

Ataque XML Entity Expansion.- No se descargan esquemas.

Billion Laughs Attack.- Se define un tamaño máximo para los ENTITIES.

Se comprueba que los esquemas recibidos son los que deben ser. Se comprueba que no hay esquemas adicionales declarados para prevenir envenenamientos.

Se valida el esquema del XML recibido. Esto implica evitar ataques de solapamiento en los que se agrega información al XML con la intención de confundir las referencias de la firma.

Para ello, durante el pre-parseo se lee el XML con los siguientes parámetros:

- factory.setNamespaceAware(true)
- factory.setValidating(true)
- Se declaran los esquemas para no descargarlos:
    - factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage", XMLConstants.W3C_XML_SCHEMA_NS_URI);
    - factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource", new InputSource(schemaFile));
- factory.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "file,jar");
  factory.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "file,jar");
- Para prevenir ataques DOS:
  factory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);

10. Glosario de términos

Assertion: Conjunto de datos empaquetados en un mensaje de SAML, se dividen en tres tipos de aserciones, de autenticación, de autorización o atributos declarados.

Http Redirect: Es un estado que emite el servidor (30X) para redirigir al navegador del cliente a una URL.

JKS: Java Key Store, es un tipo de fichero orientado a almacenar certificados y claves privadas para servidores de aplicaciones. Se suelen dividir en dos, keystore o "Privare Key entries" orientado claves privadas acompañadas por el certificado asociado a la clave pública, y truststore o "Trusted Certificate Entries" orientado almacenar las claves públicas de otras entidades.

Keytool: es una utilidad de java para la gestión y manejo de certificados (normalmente se incluye junto con la JDK).

Maven: Herramienta de gestión de proyectos de software, que entre otras funcionalidades nos aporta las dependencias y gestión de librerías necesarias.

Pom.xml: Es un fichero XML que define la información acerca del proyecto de Maven, fuentes, dependencias, test…

11. Enlaces de interés

[1] SAML, [Website], https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language

[2] OASIS, [Website], https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[3] XML, [Website], https://es.wikipedia.org/wiki/Extensible_Markup_Language