ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA SUPERIOR INFORMÁTICA

BIBLIOTECA PARA EL PERFIL DE ASERCIÓN DEL PROTOCOLO OAUTH2 E IMPLANTACIÓN EN EL SERVICIO DE IDENTIDAD DE RedIRIS

Realizado por

LUIS JAVIER GÓMEZ SANTANA

N.I.F. : 28787309-A

Dirigido por

DIEGO R. LÓPEZ

Y

MANUEL VALENCIA

Departamento

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Sevilla, Septiembre de 2012

Tabla de contenido

1.INTRODUCCIÓN 5

2. OAUTH 2.0 8

2.1 INTRODUCCIÓN 8

2.2 ANTECEDENTES A OAUTH 8

2.3 ROLES 9

2.4 FLUJO 10

2.5 CONCESIÓN DE AUTORIZACIÓN (AUTHORIZATION GRANT) 12

2.6 OAUTH 2.0 ASSERTION PROFILE(PERFIL ASERCIÓN OAUTH 2.0) 13

2.7 SAML 2.0 BEARER ASSERTION PROFILE FOR OAUTH 2.0(PERFIL DE ASERCIÓN SAML 2.0 PARA OAUTH 2.0) 14

2.7.1 SAML 2.0 15

2.7.2 PERFIL ASERCIÓN SAML 2.0 PARA OAUTH 2.0 17

2.7.2.1 USO DE ASERCIONES SAML COMO AUTHORIZATION GRANT 18

2.7.2.2 FORMATO DE LA ASERCIÓN Y REQUISITOS DE PROCESAMIENTO 19

2.7.2.3 PROCESAMIENTO DEL AUTHORIZATION GRANT 22

2.8 REGISTRO DE CLIENTES 23

2.8.1 TIPOS DE CLIENTE 23

2.8.2 IDENTIFICADOR DE CLIENTE 25

2.8.3 AUTENTICACIÓN DE CLIENTE 26

2.8.3.1 SECRETO DE CLIENTE 26

2.8.3.2 CLIENTES NO REGISTRADOS 28

2.9 EXTREMOS DEL PROTOCOLO (PROTOCOL ENDPOINTS) 28

2.9.1 TOKEN ENDPOINT 29

2.9.2 CAMPO DE APLICACIÓN DEL TOKEN DE ACCESO 29

2.10 EMITIENDO TOKENS DE ACCESO 30

2.10.1 RESPUESTA SATISFACTORIA 30

2.10.2 RESPUESTA DE ERROR 32

2.11 ACCEDIENDO A RECURSOS PROTEGIDOS 34

2.11.1 TIPOS DE TOKENS DE ACCESO 35

2.11.1.1 TOKENS PORTADORES (BEARER TOKENS) 35

2.11.1.2 OTROS TIPOS DE TOKENS 38

2.12 USO DE ASERCIONES PAPI COMO AUTHORIZATION GRANT 39

2.12.1 PAPI 39

2.12.2 ASERCIONES PAPI 41

3. ANÁLISIS DE LA APORTACIÓN REALIZADA 44

3.1 ANTECEDENTES OAUTH 2.0 EN REDIRIS 44

3.1.1 SERVICIO DE IDENTIDAD DE REDIRIS, SIR 44

3.1.1.1 CARACTERÍSTICAS DEL SERVICIO DE IDENTIDAD DE REDIRIS 49

3.1.2 INTEGRACIÓN DEL PROTOCOLO OAUTH CON EL SIR 51

3.1.3 BIBLIOTECA OAUTH2 PARA EL PERFIL DE ASERCIÓN, REDIRIS 53

3.1.4 ENTORNO SIR PARA EL PROTOCOLO OAUTH2 54

3.2 OBJETIVOS PARA LA MEJORA DE LA BIBLIOTECA 54

3.2.1 FORMATO DE TOKENS JWT(JSON WEB TOKEN) 55

3.2.2 AUTENTICACIÓN Y AUTORIZACIÓN PHPPOA EN SERVIDOR DE RECURSOS 58

3.2.3 REPASAR CLIENTE 59

3.2.4 OTROS OBJETIVOS 61

4. ANÁLISIS DE REQUISITOS, DISEÑO E IMPLEMENTACIÓN 63

4.1 CLIENTE 63

4.2 SERVIDOR DE AUTORIZACIÓN(AS) 66

4.3 SERVIDOR DE RECURSOS(RS) 70

5. ANÁLISIS TEMPORAL Y DE COSTES 73

5.1 ESTIMACIONES INICIALES 73

5.1 ESTIMACIONES FINALES 75

6. MANUAL DE USUARIO 79

6.1 ARCHIVOS DE CONFIGURACIÓN 79

6.1.1 CLIENTE 79

6.1.2 SERVIDOR DE AUTORIZACIÓN (AS) 83

6.1.3 SERVIDOR DE RECURSOS(RS) 88

6.2 USO DEL LA BIBLIOTECA OAUTH2LIB 90

7. CONCLUSIONES 93

BIBLIOGRAFÍA 96

GLOSARIO DE TÉRMINOS 98

1.INTRODUCCIÓN

Este proyecto, “Biblioteca para el perfil de aserción del protocolo OAuth2 e implantación en el servicio de identidad de RedIRIS”, trata principalmente sobre el problema de la autorización, por parte de usuarios previamente autenticados, de acceso a recursos que están bajo su control, llevada a cabo de una manera segura y controlada, y sobre todo, sin que se vean comprometidas las credenciales del usuario. Todo este proceso se realiza en entornos cliente-servidor, propios de la interacción de aplicaciones web o de escritorio, con servicios web.

La idea principal sobre la que subyace el interés por desarrollar una herramienta que sea capaz de ofrecer un servicio para dar solución al problema antes expuesto se puede ver en el siguiente ejemplo: una aplicación “nueva” incluye determinado servicio, el cual deberá de importar de otra aplicación. Para poder activar este servicio, es necesario que el usuario en cuestión tenga que ofrecer, a la aplicación nueva, su nombre de usuario y contraseña(o, en general, sus credenciales personales) para que esta pueda acceder a la información necesaria para poder ofrecer el servicio. El resultado obtenido es que ambas aplicaciones funcionan correctamente, salvo por un inconveniente, y es que la aplicación nueva que ofrece

el servicio ya posee las credenciales personales del usuario, lo cual puede acarrear un serio problema de seguridad.

Es por ello que surgió la necesidad de establecer un protocolo o estándar de carácter abierto que diera solución a ese problema, restringiendo el acceso a los recursos. Este protocolo se llama OAuth(Open Authorization, Autorización abierta), que actualmente se encuentra en la versión 2.0 (todavía como borrador).

Durante 2010, el departamento de middleware de RedIRIS implementó una biblioteca, escrita en lenguaje PHP, para el protocolo OAuth2, la cual será motivo de estudio para su mejora en la realización de este proyecto. Básicamente, esta biblioteca está diseñada, principalmente, para que pueda ser usada por las instituciones afiliadas al Servicio de Identidad de RedIRIS (SIR), permitiéndoles, a través de una aplicación cliente, poder acceder a recursos protegidos dentro de un *Servidor de Recursos*(RS).

Para ello, la aplicación cliente, previamente registrada en un *Servidor de Autorización*(AS), solicitará a este un *token de acceso*, mandando en la petición una serie de credenciales necesarias para la obtención del token. El AS, una vez comprobada la validez de las credenciales recibidas, procederá a enviar al cliente el *token de acceso*, que podrá usarlo para hacer la petición de un recurso al *Servidor de Recursos*(RS). Todo este proceso tiene como guía al protocolo OAuth2, descrito anteriormente. El estudio en profundidad de este protocolo se realizará en el *capítulo 2 OAUTH 2.0*.

Para la realización de este proyecto se partió de la base de una biblioteca ya implementada por desarrolladores del SIR, concretamente la versión *OAuth2lib_v0.12*, la cual será estudiada en el capítulo 3, referente al estudio de los antecedentes y de la aportación realizada, donde se verá, además del modelo de implantación del protocolo OAuth 2.0 en el SIR, un detalle de los objetivos planteados para la modificación y mejora de la biblioteca *OAuth2lib_v0.12*.

En el capítulo 4 se realizará un análisis del diseño y la implementación de la biblioteca. Este análisis se realizará por partes, describiendo la estructura de cada parte de la biblioteca: *Cliente, Servidor de Autorización y Servidor de Recursos*.

Además, este documento contiene un capítulo(capítulo 5) en el que se comparan los esfuerzos estimados inicialmente con los esfuerzos requeridos para desarrollar el proyecto.

Por último, se verá un capítulo con el Manual de Usuario de la biblioteca(capitulo 6), donde se explican con todo detalle los mecanismos de instalación, configuración e implantación tanto del cliente como de los servidores(de autorización y de recursos), además de un capítulo de conclusiones donde se analizan las decisiones tomadas durante la realización de este proyecto y la proyección de futuro que se le pretende dar al mismo.

2. OAUTH 2.0

2.1 INTRODUCCIÓN

OAuth es un protocolo de autorización, que actualmente se encuentra en versión draft en la organización IETF(Internet Engineering Task Force), y que, básicamente, permite a una aplicación cliente acceder y procesar los recursos de un usuario en su nombre, siempre que el propietario del recurso haya dado su consentimiento. Para ello, habilita a una aplicación hecha por terceros a tener acceso limitado a un servicio HTTP, ya sea en nombre del propietario de un recurso (haciendo uso del consentimiento dado por parte del mismo al propio servicio HTTP), o permitiendo a la aplicación de terceros obtener acceso por sí sola, sin necesidad del consentimiento del propietario (debe haber establecida una relación de confianza entre la aplicación de terceros y el propietario del recurso).

2.2 ANTECEDENTES A OAUTH

En el modelo tradicional de autenticación cliente-servidor, el cliente pedía al servidor acceso a un recurso protegido, y para ello debía de autenticarse en el servidor usando las credenciales del propietario del recurso(por ejemplo, nombre

de usuario y contraseña). Esto creaba diversos problemas de seguridad y ciertas limitaciones:

- Las aplicaciones de terceros necesitaban guardar las credenciales del propietario del recurso para posibles usos en un futuro, típicamente un nombre de usuario y una contraseña guardada en texto plano.
- Los servidores necesitaban soportar autenticación vía password, a pesar de los problemas de seguridad que esto mismo supone.
- Las aplicaciones de terceros obtenían un acceso demasiado amplio a los recursos protegidos, dejando a su propietario sin ninguna capacidad para restringir la duración de ese acceso, limitar el acceso a sólo un subconjunto de recursos o incluso poder revocar esa acceso por parte de la aplicación a sus recursos.

OAuth ataca estos problemas/limitaciones introduciendo una capa de *autorización* en todo este proceso, y separando el rol que tiene el cliente del rol del propietario del recurso. En OAuth, el cliente solicita acceso a los recursos, controlados por el propietario del recurso y almacenado en un Servidor de Recursos, usando para ello unas credenciales no tan comprometedoras para el propietario del recurso.

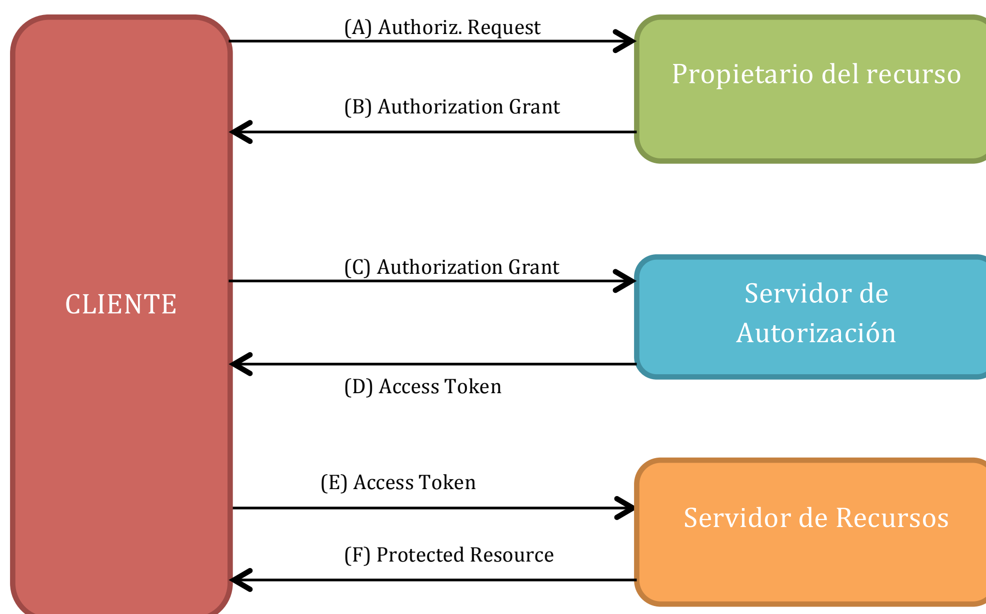
2.3 ROLES

- ***Propietario del recurso*** : entidad capacitada para conceder acceso a un recurso protegido.

- ***Servidor de Recurso*** : el servidor que aloja los recursos protegidos, siendo capaz de aceptar y responder a las solicitudes de recursos que le llegan de un cliente.
- ***Cliente*** : una aplicación capaz de hacer peticiones a recursos protegidos, en nombre del propietario del recurso, y con su autorización.
- ***Servidor de Autorización*** : es el servidor encargado de emitir *tokens de acceso* a clientes, los cuales han tenido que obtener previamente autorización del propietario del recurso.

2.4 FLUJO

A continuación se muestra el flujo general para la obtención de un recurso mediante OAuth 2.0:



(A) El cliente solicita la autorización del propietario del recurso. La petición de autorización puede ser hecha directamente al propietario del recurso (como se muestra), o de manera indirecta, la cual es preferible, usando como intermediario el servidor de autorización

(B) El cliente recibe una concesión de autorización(*authorization grant*), la cual representa la autorización del propietario del recurso. El tipo de esta concesión depende del método usado por el cliente para realizar la petición y de los tipos soportados por el servidor de autorización(se verá más adelante).

(C) El Cliente solicita un *token de acceso* mediante la autenticación en el Servidor de Autorización presentando el *authorization grant*.

(D) El Servidor de Autorización autentica al cliente y valida el *authorization grant*, y si es válido emite un *token de acceso.*

(E) El Cliente realiza una petición de un recurso protegido al Servidor de Recursos presentando el *token de acceso*.

(F) El Servidor de Recursos valida el *token de acceso* recibido, y si es válido, sirve la solicitud.

2.5 CONCESIÓN DE AUTORIZACIÓN (AUTHORIZATION GRANT)

Un *authorization grant* no es más que una credencial que representa la autorización dada por el propietario del recurso para que se pueda acceder a sus recursos, la cual va a ser usada por el cliente para obtener un token de acceso.

OAuth 2.0 define, principalmente, 4 formas en las que la aplicación cliente puede obtener autorización por parte del propietario del recurso. Estas son:

- ***Authorization Code*** – se obtiene a través de un servidor de autorización, que hace de intermediario entre el cliente y el propietario del recurso.
- ***Implicit*** – es una versión simplificada del anterior, optimizada para entornos que usen lenguajes de script, capaces de ejecutar programas en una página web al ser visualizados en un navegador de internet, como pueda ser JavaScript.
- ***Resource Owner Password Credentials*** – se usan las credenciales del propietario del recurso (por ejemplo, nombre de usuario y password) para obtener el authorization grant, siempre y cuando exista una relación de confianza entre el cliente y el propietario del recurso.
- ***Client Credentials*** – las credenciales del cliente pueden ser usadas como authorization grant cuando el alcance de la autorización está limitado a los recursos protegidos que están bajo el control del cliente, o estos han sido previamente establecidos con un Servidor de Autorización.

A parte de esas cuatro formas de conseguir un *authorization grant*, OAuth permite un mecanismo de extensión por el cual se pueden definir nuevos métodos para obtener un authorization grant. Uno de estos métodos, denominado perfil de aserción de OAuth 2.0, es el método por el cual se va a obtener el *authorization grant* en esta biblioteca.

2.6 OAUTH 2.0 ASSERTION PROFILE(PERFIL ASERCIÓN OAUTH 2.0)

OAuth 2.0 provee una especificación general para usar una aserción como credenciales de cliente para obtener un *authorization grant*, o para que la propia aserción sirva como *authorization grant*(este es el uso que se va a dar en la biblioteca a las aserciones). La intención de este perfil es mejorar la seguridad usando firmas digitales y métodos criptográficos para garantizar la autenticidad de los datos enviados desde un cliente, además de facilitar la integración de OAuth 2.0 en escenarios cliente-servidor donde el usuario final puede no estar presente.

Una aserción no es más que una serie de atributos de usuario, que suelen llegar en forma de xml(SAML) o de cadena de texto(PAPI), y que son emitidas por un proveedor de identidad (IdP, Identity Provider) para que sean consumidas/utilizadas por un proveedor de servicios(SP, Service Provider).

Esta especificación es un poco general, sólo da ideas de los mecanismos genéricos para el transporte de las aserciones en la interacción del cliente con el Servidor de Autorización. La biblioteca que se ha implementado va a usar 2 tipos de aserciones,

PAPI y SAML 2.0 como *authorization grant*(recordemos que se podía usar la aserción como credenciales de cliente para obtener un *authorization grant* y como el propio authorization grant).

Debido a que no existe actualmente un perfil específico que de soporte a aserciones PAPI dentro de OAuth 2.0, se ha usado esta especificación general como guía para el uso de aserciones tipo PAPI.

Actualmente este perfil es demasiado general, por lo que se han creado otros perfiles más específicos(siempre dentro del WG de OAuth 2.0 del IETF). En la siguiente sección se pasará a explicar detalladamente el perfil SAML 2.0 Assertion Profile, que es el que más se ha tenido en cuenta a la hora de implementar la biblioteca.

2.7 SAML 2.0 BEARER ASSERTION PROFILE FOR OAUTH 2.0(PERFIL DE ASERCIÓN SAML 2.0 PARA OAUTH 2.0)

Antes de pasar a definir las características del perfil de aserción SAML 2.0 para OAuth 2.0, hay que conocer qué es SAML 2.0 y qué forma tiene una aserción de este tipo.

2.7.1 SAML 2.0

SAML 2.0, cuyas siglas significan Security Assertion Markup Language, o Lenguaje de Marcado de Aserciones de Seguridad, es la segunda versión del estándar propuesto por OASIS(Organization for the Advancement of Structured Information Standards) para el intercambio de datos de autenticación y autorización entre dominios de seguridad. Su primera intención fue ofrecer una especificación dirigida al dominio de navegadores web para ofrecer un servicio de inicio de sesión único(Single Sign-on), aunque también fue diseñada de forma modular y extensible para facilitar su uso en otros contextos.

Se trata de un protocolo basado en XML que se usa en la comunicación entre un Proveedor de Identidad (IdP) y un Servicio Web(Web Service, WS). La aserción, que se puede denominar como un token de seguridad XML, es lo que se emite desde el IdP y que es consumido por el WS, el cual debe de confiar en el contenido de la aserción, más concretamente en su *subject*, por motivos relacionados con la seguridad.

La siguiente figura muestra un ejemplo de una aserción SAML 2.0 muy similar a la que se usará en este proyecto como Authorization Grant:

```
1. <Assertion IssueInstant="2012-03-01T20:59:34.619Z"
2. ID="ef1xs3RxIP6oqjd7HTLRLIBlBb7"
3. Version="2.0"
4. xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
5. <Issuer>https://samlExample.idp.com</Issuer>
6. <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
7. [omitido por brevedad]
8. </ds:Signature>
9. <Subject>
10. <NameID
11. Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
12. example@idp.com
13. </NameID>
14. <SubjectConfirmation
15. Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
16. <SubjectConfirmationData
17. NotOnOrAfter="2012-10-01T20:12:34.619Z"
18. Recipient="https://authz.idp.org/token.oauth2"/>
19. </SubjectConfirmation>
20. </Subject>
21. <Conditions>
22. <AudienceRestriction>
23. <Audience>https://saml-example.sp.net</Audience>
24. </AudienceRestriction>
25. </Conditions>
26. <AuthnStatement AuthnInstant="2012-10-01T20:07:34.371Z">
27. <AuthnContext>
28. <AuthnContextClassRef>
29. urn:oasis:names:tc:SAML:2.0:ac:classes:X509
30. </AuthnContextClassRef>
31. </AuthnContext>
32. </AuthnStatement>
33. </Assertion>
```

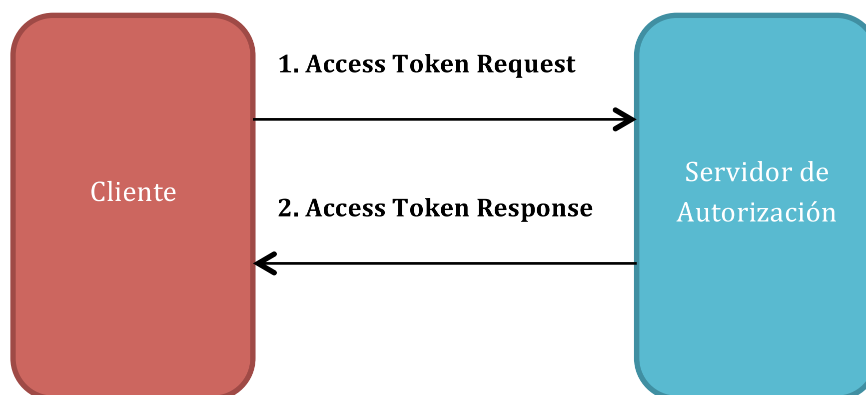
Los elementos más relevantes son los siguientes:

- ***Issuer*** : Es la autoridad SAML que ha emitido la aserción. No necesariamente tiene que ser la misma entidad que firma la aserción (elemento ***Signature***).
- ***Subject*** : Es el sujeto al que hace referencia la aserción.

- ***SubjectConfirmation*** : Información que permite al sujeto ser confirmado. En caso de que exista más de un elemento *SubjectConfirmation*, es suficiente con satisfacer uno de ellos para confirmar al sujeto de la aserción.
- ***Conditions*** : Son condiciones que se deben evaluar a la hora de comprobar la validez de la aserción. Puede contener a los elementos ***AudienceRestriction***, para indicar que la aserción se redirecciona a una o más audiencias particulares, identificadas por los elementos ***Audience***.

2.7.2 PERFIL ASERCIÓN SAML 2.0 PARA OAUTH 2.0

A continuación se definirá cómo una aserción SAML puede ser usada para pedir un token de acceso (al Servidor de Autorización), siempre en el contexto de que el cliente quiere hacer uso de la existencia de una **relación de confianza** (como en entornos federados), expresada a través de la semántica de la aserción SAML, sin necesidad de la autorización expresa del propietario del recurso al Servidor de Autorización, haciendo que el flujo natural de OAuth se salte el paso de la obtención del *Authorization Grant*, ya que, como se comentó anteriormente, la propia aserción SAML se usará como *Authorization Grant*.



1. El cliente realiza una petición de un *token de acceso* al *Servidor de Autorización* presentando una aserción SAML, la cual actuará como garante de la autorización del propietario del recurso, y unas credenciales de cliente.
2. El *Servidor de Autorización* verifica la validez de la aserción, enviando de vuelta al cliente un *token de acceso* si no ha habido problemas, o una respuesta de error si la aserción no es válida.

2.7.2.1 USO DE ASERCIONES SAML COMO AUTHORIZATION GRANT

Para usar aserciones SAML como *authorization grant*, es necesario usar los siguientes parámetros y codificaciones:

- El valor del parámetro **grant_type**(enviado dentro de la petición) debe ser el siguiente:

  "urn:ietf:params:oauth:grant-type-saml2-bearer"

- El valor de la aserción(enviada dentro de la petición) debe contener una aserción SAML 2.0. Los datos XML de la aserción SAML deben estar codificados usando base64url.

2.7.2.2 FORMATO DE LA ASERCIÓN Y REQUISITOS DE PROCESAMIENTO

Con el fin de emitir una respuesta con un *token de acceso*, el *servidor de autorización* debe validar la aserción de acuerdo con el siguiente criterio:

- El elemento **<Issuer>** de la aserción debe contener un identificador único de la entidad que emitió la aserción.
- La aserción debe contener un elemento **<Conditions>,** el cual debe contener un elemento **<AudienceRestriction>**, que a su vez deberá contener un elemento **<Audience>**, cuyo valor debe ser una referencia URI que identifique inequívocamente al Servidor de Autorización, o a un proveedor de servicios SAML en el dominio de control del Servidor de Autorización al cual se realiza la petición. La URL del *token endpoint (*este concepto se definirá más adelante*)* del Servidor de Autorización puede ser usada como valor para el elemento **<Audience>**. El Servidor de Autorización deberá verificar la validez de este elemento.
- La aserción debe contener un elemento **<Subject>**, el cual puede identificar al propietario del recurso para el que se está realizando la petición del *token de acceso*. En el caso del uso de aserciones SAML como *authorization grant*, el **<Subject>** debería identificar una entidad autorizada(típicamente al propietario del recurso, o una entidad delegada autorizada). Información adicional para identificar al sujeto de la transacción puede ser incluido en el elemento **<AttributeStatement>**.

- La aserción debe tener un tiempo de expiración que limite la ventana de tiempo durante la cual puede ser usada. Este tiempo de expiración puede estar en dos partes de la aserción:
  - En el atributo *NotOnOrAfter* del elemento **<Conditions>**
  - En el atributo NotOnOrAfter del elemento **<SubjectConfirmationData>**.
- El elemento **<Subject>** debe contener al menos un **<SubjectConfirmation>** que permita al Servidor de Autorización confirmar la aserción como de tipo *bearer* (mediante el atributo method, que tendrá el valor "urn:oasis:names:tc:SAML:2.0:cm:bearer"). Además, el elemento **<SubjectConfirmation>** debe contener un elemento **<SubjectConfirmationData>**, a menos que la aserción contenga un elemento **<Conditions>** con un atributo *NotOnOrAfter* válido, en cuyo caso el elemento **<SubjectConfirmationData>** puede ser omitido. Si está presente, este elemento debe tener un atributo llamado Recipient cuyo valor tiene que indicar la URL del *token endpoint* del Servidor de Autorización. Este debe de verificar que el valor del atributo Recipient concuerde con la URL del *token endpoint*(o un alias aceptable) para la cual fue entregada la aserción. El elemento **<SubjectConfirmationData>** debe tener un atributo *NotOnOrAfter* que limite la ventana de tiempo en la cual la aserción puede ser confirmada. También puede contener un atributo Address que limite la dirección desde la cual la aserción puede ser

entregada. La verificación de esta dirección queda a discreción del *Servidor de Autorización*.

- El *Servidor de Autorización* debe verificar que las ocurrencias del atributo *NotOnOrAfter* no hayan pasado todavía. Un tiempo expirado en el atributo *NotOnOrAfter* en el elemento **<Conditions>** invalida por completo la aserción. Si el tiempo expirado está en el atributo *NotOnOrAfter* del elemento **<SubjectConfirmationData>** sólo invalida el elemento **<SubjectConfirmation>** al que pertenece. Por motivos de seguridad, el *Servidor de Autorización* puede rechazar peticiones con un instante *NotOnOrAfter* que este razonablemente lejos en el tiempo.
- Si el emisor de la aserción autentica al **<Subject>**, la aserción debería contener un elemento **<AuthnStatement>** que represente el evento de autenticación.
- Si la aserción fue emitida con la intención de que el *Presenter* actúa de manera autónoma en nombre del *Subject*, el elemento **<AuthnStatement>** no debería ser incluido. El *Presenter* debe ser identificado en un elemento **<NameID>**, o en el elemento **<SubjectConfirmation>**.
- Otras declaraciones, en particular los elementos **<AttributeStatements>**, pueden ser incluidos en la aserción.
- La aserción debe estar firmada digitalmente por el emisor, y el *Servidor de Autorización* está obligado a verificar esa firma.

2.7.2.3 PROCESAMIENTO DEL AUTHORIZATION GRANT

El *Servidor de Autorización* debe también validar las credenciales del cliente, las cuales no se deben confundir con las del *propietario del recurso*(las credenciales del propietario del recurso no son conocidas en ningún momento), son las credenciales de la aplicación cliente(normalmente un identificador de cliente y un secreto de cliente). Además, el *Servidor de Autorización* debería de emitir *tokens de acceso* con un tiempo de vida limitado, que en caso de perder la validez obligará al cliente a solicitar un nuevo *token de acceso* al *Servidor de Autorización*, presentando la misma aserción(en caso de ser todavía válida), o con una nueva.

Si la aserción no es válida(atributo *NotOnOrAfter* no válido), o si los requerimientos del elemento **<SubjectConfirmation>** no se pueden cumplir, el *Servidor de Autorización* debe responder con un mensaje de error siguiendo la estructura de mensajes de error usada en OAuth 2.0. Esta estructura es la siguiente:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
{
    “error” : “invalid_grant”,
    “error_description” : “Audience validation failed”
}
```

2.8 REGISTRO DE CLIENTES

Antes de iniciar el protocolo, el cliente tiene que estar registrado en el *Servidor de Autorización*. Los medios por los cuales el cliente se registra en el *Servidor de Autorización* están fuera del alcance de esta especificación.

El registro de clientes en *Servidores de Autorización* no requiere una interacción directa entre el Cliente y el *Servidor de Autorización*. Si el registro es soportado por el *Servidor de Autorización*, este puede recurrir a otros medios para establecer la relación de confianza y la obtención de las propiedades requeridas del cliente(URI de redirección, tipo de cliente, *authorization grant* que se va a usar,...).

Cuando se registra un cliente, el propietario o administrador de ese cliente debe:

- Especificar el tipo de cliente
- Ofrecer una URI para la redirección del cliente
- Incluir otro tipo de información que sea requerida por el *Servidor de Autorización*, por ejemplo: nombre de la aplicación, descripción, el consentimiento a términos legales, ...

2.8.1 TIPOS DE CLIENTE

OAuth define dos tipos de cliente, basados en su habilidad para autenticarse de manera segura con el *Servidor de Autorización*.

- ***Confidencial*** : son los clientes capaces de mantener la confidencialidad de sus credenciales, por ejemplo, un cliente implementado en un servidor seguro con acceso restringido a las credenciales del cliente. También puede ser capaz de garantizar la seguridad de las credenciales por otros medios.
- ***Público*** : son clientes incapaces de mantener la confidencialidad de sus credenciales, por ejemplo, un cliente que se ejecuta en un dispositivo del propietario del recurso, como puede ser una aplicación nativa instalado o una aplicación web basada en navegador.

La designación del tipo de cliente se basa en la definición de autenticación segura que se haya implementado en el *Servidor de Autorización* y en los niveles de exposición de las credenciales del cliente.

OAuth ha sido diseñado en torno a los siguientes perfiles(hay que diferenciar entre tipo y perfil) de cliente:

- ***Aplicación Web*** : una aplicación web es un cliente confidencial que es ejecutado en un *servidor web*. Los propietarios de recursos acceden al cliente a través de una interfaz de usuario HTML, representada en un *user-agent*(agente de usuario*)* en el dispositivo *del propietario del recurso*. Las credenciales del cliente, así como cualquier *token de acceso* emitido al cliente son guardados en el servidor web y no son accesibles por el propietario del recurso.
- ***Aplicación basada en user-agent*** : una aplicación de usuario basada en user-agent es un cliente público en el que el código del cliente se descarga

de un *servidor web* y se ejecuta dentro de un agente de usuario(user-agent) en el dispositivo del propietario del recurso. Los datos y credenciales del protocolo son fácilmente accesibles(y usualmente visibles) al propietario del recurso. Dado que estas aplicaciones residen en el agente de usuario del propietario del recurso, pueden hacer un uso transparente de las capacidades del agente de usuario al solicitar la autorización.

- ***Aplicaciones Nativas*** : Una aplicación nativa es un cliente público instalado y ejecutado en el dispositivo del propietario del recurso. Los datos y credenciales del protocolo son accesibles por el propietario del recurso. Se asume que cualquier credencial de autenticación del cliente incluida en la aplicación puede ser extraída. Por otro lado, las credenciales emitidas dinámicamente, tales como el *token de acceso*, sí tienen un nivel aceptable de protección, garantizando como mínimo que esas credenciales están protegidas de servidores hostiles que puedan interactuar con la aplicación, o de aplicaciones dentro del mismo dispositivo.

2.8.2 IDENTIFICADOR DE CLIENTE

El *Servidor de Autorización* otorga al cliente, durante el proceso de registro, un identificador de cliente, que no es secreto, está expuesto al propietario del recurso, y no debe ser usado en solitario para la autenticación del cliente.

2.8.3 AUTENTICACIÓN DE CLIENTE

Si el cliente es de tipo *confidencial*, el *Servidor de Autorización* y el *Cliente* deben establecer un método de autenticación adecuado a los requerimientos de seguridad del *Servidor de Autorización*. Este puede aceptar cualquier tipo de autenticación de clientes siempre que concuerde con los requerimientos de seguridad.

Los clientes *confidenciales* suelen establecer una serie de credenciales de cliente que se usan para la autenticación con el *Servidor de Autorización*.

El *Servidor de Autorización* no debe de hacer suposiciones sobre el tipo de cliente o aceptar el tipo de información recibida sin haber establecido previamente una relación de confianza con el cliente o su desarrollador. El *Servidor de Autorización* puede establecer métodos de autenticación con clientes de tipo *público*. Sin embargo, no debería confiar en autenticaciones de clientes públicos con el propósito de identificar al cliente.

El cliente no puede usar más de un método de autenticación en cada petición.

2.8.3.1 SECRETO DE CLIENTE

Los *Clientes* en posesión de un *secreto de cliente (client_secret)*, deben usar el esquema de autenticación HTTP Basic (definido en RFC2627). Según este

esquema, se debe de incluir en la petición del *token de acceso* al *Servidor de Autorización* una cabecera HTTP con el siguiente prototipo:

Authorization : Basic <VALOR CIFRADO>

Siendo *<VALOR CIFRADO>* el valor del identificador del cliente cifrado mediante el método *HMAC*(con algoritmo para cifrar sha256), usando como clave el *secreto de cliente*.

De forma alternativa, el *Servidor de Autorización* puede permitir la inclusión de las credenciales del cliente en el cuerpo(body) de la petición, pero está práctica es altamente desaconsejable debido a los problemas de seguridad y suplantación de la identidad de clientes que puede conllevar.

El *Servidor de Autorización* debe exigir el uso por parte del cliente de mecanismos de seguridad en la capa de transporte (*SSL* y *TLS*) al enviar las peticiones de *tokens de acceso*, ya que sino podrían extraerse de la transmisión las credenciales del cliente en texto en claro. También debe protegerse de *ataques de fuerza bruta* que traten de obtener el *secreto del cliente*.

El formato de las peticiones, tanto de *tokens de acceso* como de *recursos compartidos*, así como los detalles de la implementación de la biblioteca implementada en este proyecto serán ampliamente discutidos en posteriores capítulos.

2.8.3.2 CLIENTES NO REGISTRADOS

La especificación de OAuth 2.0 no excluye el uso de clientes no registrados. Sin embargo, el uso de este tipo de clientes queda fuera del alcance de esta especificación, ya que requiere de análisis de seguridad adicionales y de la revisión de su impacto interoperacional.

2.9 EXTREMOS DEL PROTOCOLO (PROTOCOL ENDPOINTS)

El proceso de autorización utiliza dos puntos de entrada(*endpoints*) al flujo de ejecución de un proceso de petición de recursos mediante OAuth. Estos son:

- ***Authorization Endpoint*** : usado para obtener autorización del propietario del recurso a través de una redirección del agente de usuario.
- ***Token Endpoint*** : usado para el intercambio de un *authorization grant* por un *token de acceso* durante la operación de autenticación de un cliente.

No todos los *authorization grants* utilizan ambos tipos de *endpoints*. Los tipos de concesión extendidos (extension grant types) pueden definir *endpoints* adicionales según necesiten.

Un ejemplo de ello se encuentra en la biblioteca implementada para este proyecto, que no usa el *authorization endpoint* debido a que la autorización del propietario del recurso viene implícita dentro de la aserción enviada en la petición del *token de*

acceso. Además se usa otro *endpoint* , denominado *Server Endpoint*, incluido en el *Servidor de Recursos* y usado para el intercambio de un *token de acceso* por un recurso protegido.

2.9.1 TOKEN ENDPOINT

El *token endpoint* es usado por el cliente para obtener un *token de acceso*, presentado para ello un *authorization grant* o un *token de refresco*. El *token endpoint* es usado por todos los tipos de *authorization grant*, excepto por el tipo implícito(Implicit Grant).

El medio por el cual el cliente obtiene la localización(URL) del *token endpoint* está fuera del alcance de la especificación de OAuth 2.0.

2.9.2 CAMPO DE APLICACIÓN DEL TOKEN DE ACCESO

El campo de aplicación(a partir de ahora se usara el término anglosajón scope para referirse a este ámbito de aplicación) del *token de acceso* es un parámetro que se ha de enviar al *token endpoint*(también al *authorization endpoint* en caso de existir) para permitir que el cliente pueda especificar la amplitud de la petición de acceso, entendiendo por amplitud el número de recursos a los que es capaz de acceder. A su vez, el *Servidor de Autorización* usa el parámetro *scope* que incluye en la respuesta al cliente para informarle del alcance del *token de acceso* emitido.

El valor del parámetro *scope* se ha de expresar como una lista de cadenas de caracteres separada por espacios en blanco y con diferenciación entre mayúsculas y minúsculas. Estas cadenas han de ser definidas por el *Servidor de Autorización*. Si el valor contiene múltiples cadenas separadas por espacios en blanco, el orden de estas cadenas no es relevante, y cada cadena añade un rango de acceso adicional al *scope* requerido.

El *Servidor de Autorización* puede ignorar total o parcialmente el *scope* requerido por el cliente, basándose en las políticas implementadas en el *Servidor de Autorización*.

2.10 EMITIENDO TOKENS DE ACCESO

Si la petición del *token de acceso* es válida y está autorizada, el *Servidor de Autorización* ha de emitir un *token de acceso* en una respuesta satisfactoria, tal como se va a debatir a continuación. En cambio, si en la petición falla la operación de autenticación del cliente o si es inválida, el *Servidor de Autorización* debe de enviar una respuesta de error al cliente.

2.10.1 RESPUESTA SATISFACTORIA

El *Servidor de Autorización* emite un *token de acceso* construyendo la respuesta añadiendo los siguientes parámetros al *cuerpo*(body) de la respuesta HTTP:

- ***access _token*** : REQUERIDO. El token de acceso emitido por el *Servidor de Autorización.*
- ***token_type*** : REQUERIDO. El tipo de token emitido. Los tipos de *token de acceso* serán estudiados en apartados posteriores. Su valor es sensible a mayúsculas y minúsculas.
- ***expires_in*** : OPCIONAL. El tiempo de vida en segundos del *token de acceso*. Por ejemplo, un valor de 3600 en este parámetro indica que el *token de acceso* expirará en una hora desde que la respuesta fue emitida.
- ***scope*** : OPCIONAL. El campo de aplicación o ámbito del *token de acceso*.

Estos parámetros han de ser incluidos en el *body* de la respuesta http que emite el *Servidor de Autorización* usando para ello el tipo de medios *application/json*. Para ello, los parámetros son serializados en una estructura de tipo JSON, añadiendo cada parámetro en el nivel más alto de la estructura. Los parámetros cuyo valor son cadenas de caracteres son incluidos como cadenas JSON(*strings*), así como los valores numéricos serán expresados como números en JSON. El orden de los parámetros no es relevante y puede variar según la respuesta.

El *Servidor de Autorización* debe incluir las siguientes cabeceras HTTP en cualquier respuesta que contenga un *token de acceso*, credenciales de cliente u otra información sensible:

Cache-Control : no-store

Pragma: no-cache

Así, un ejemplo de una respuesta satisfactoria emitida por un *Servidor de Autorización* tendría en siguiente formato:

HTTP/1.1 200 OK

Content-Type : Application/json: charset = UTF-8

Cache-Control: no-store

Pragma: no-cache

{

“access_token” : “3423kjhjkKJH786jHJIUiuohj”,

“token_type” : “ejemplo”,

“expires_in” : “3600”,

”example_parameter” : “valor_de_ejemplo”

}

El cliente debe de ignorar los parámetros que no reconoce de la respuesta emitida por el *Servidor de Autorización*. El tamaño de los *tokens de acceso* y de otros parámetro emitidos en la respuesta se han dejado sin definir, ya que es el propio *Servidor de Autorización* el encargado de definir estos tamaños.

2.10.2 RESPUESTA DE ERROR

El *Servidor de Autorización* debe de responder con el código de estatus http 400(Bad request, petición errónea) incluyendo los siguientes parámetros en la respuesta:

- ***error*** : REQUERIDO. Un código de error de entre los siguientes
  - ***invalid_request*** : La petición carece de algún parámetro requerido, incluyendo valores no soportados de algún parámetro, repite alguno, incluye múltiples credenciales, utiliza más de un mecanismo para autenticar al cliente, o esta mal formada en algún sentido.

  - ***invalid_client*** : La autenticación del cliente ha fallado(por ejemplo, porque el cliente no es conocido, o por que el método de autenticación no es soportado por el servidor). El *Servidor de Autorización* puede devolver un código de estatus HTTP 401(no autorizado) en vez de HTTP 400
  - ***invalid_grant*** : El *authorization grant* proporcionado por el cliente es inválido, está expirado, ha sido revocado, no concuerda la redirección URI usada en la petición de autorización o ha sido emitida para un cliente distinto al que hace la petición.
  - ***unauthorized_client*** : El cliente autenticado no está autorizado para utilizar el tipo de *authorization grant* requerido.
  - ***unsupported_grant_type*** : El tipo de *authorization grant* no es soportado por el *Servidor de Autorización*.
  - ***invalid_scope*** : El alcance solicitado es inválido, desconocido, está mal formado o excede los límites impuestos por el propietario del recurso.
- ***error_description*** : OPCIONAL. Un código leíble por personas, codificado en UTF-8, y que proporciona información adicional sobre el error, proporcionando asistencia al desarrollador del cliente sobre el tipo de error que se ha producido.
- ***error_uri*** : OPCIONAL. Una URI que identifica una página web con información acerca del tipo de error.

Así, un ejemplo de respuesta de error emitida por un *Servidor de Autorización* tendría en siguiente formato:

HTTP/1.1 400 Bad Request

Content-Type: application/json: charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"error" : "invalid_request"

}

2.11 ACCEDIENDO A RECURSOS PROTEGIDOS

El tipo de *token de acceso* proporciona al cliente la información requerida para utilizarlo de manera satisfactoria a la hora de realizar la petición de un recurso al *Servidor de Recursos*. El *Servidor de Recursos* debe validar el token recibido y asegurarse de que no está expirado y de que el *scope* cubre al recurso solicitado. El método por el cual el *Servidor de Recurso* valida un token de acceso está fuera del alcance de la especificación de OAuth 2.0, pero normalmente conlleva una interacción entre el *Servidor de Autorización* y el *Servidor de Recursos*.

La forma en la que el cliente utiliza el *token de acceso* para autenticarse con el *Servidor de Recursos* depende del tipo de token emitido por el *Servidor de Autorización*. Normalmente involucra el uso de la cabecera HTTP Authorization, usando un esquema de autenticación definido para cada tipo de token.

2.11.1 TIPOS DE TOKENS DE ACCESO

El tipo de *token de acceso* proporciona al cliente la información necesaria para utilizarlo de manera satisfactoria a la hora de realizar una petición al *Servidor de Recursos* de un recurso protegido. El cliente no debe de usar *tokens de acceso* si no comprende o confía en el tipo del token proporcionado.

En la implementación de esta biblioteca, el tipo de *token de acceso* utilizado es el definido en draft-ietf-oauth-v2-bearer-16, y que se pasa a explicar a continuación.

2.11.1.1 TOKENS PORTADORES (BEARER TOKENS)

Un token de portador(a partir de ahora se empleará el término *bearer token*) es un token de seguridad con la propiedad de que cualquier entidad(en nuestro caso esta entidad serían un cliente) en posesión de un token(un portador, *bearer*) puede usarlo de una manera única, es decir, ningún otro *bearer* podrá usar ese mismo token de la misma manera. El uso de *bearer tokens* debe de implicar la utilización de mecanismos de transporte seguro y de cifrado de datos.

Existen tres métodos por los cuales un cliente debe de ser capaz de enviar un *bearer token* en la petición al *Servidor de Recursos* de un *recurso compartido*, no pudiendo usar más de uno en cada petición. Estos tres métodos son:

1. ***Authorization Request Header Field***

Cuando se envía el token en el campo "*Authorization*" de la cabecera HTTP de la petición del recurso, el cliente está haciendo uso del esquema de autenticación *Bearer* para transmitir el token de acceso.

Por ejemplo,

GET /recurso/1 HTTP/1.1
Host : ejemplo.com
Authorization : Bearer <TOKEN DE ACCESO>

Donde *<TOKEN DE ACCESO>* representa el *token de acceso* obtenido del *Servidor de Autorización*. Debe de estar codificado en base64.

Los *Servidores de Recursos* están obligados a soportar este método de envío de tokens, siendo el método más recomendado a usar.

2. *Form-Encoded-Body Parameter*

También es posible enviar el *token de acceso* en el cuerpo de la petición HTTP. Para ello, habría que añadir el valor del token al parámetro "a*ccess_token*" del cuerpo(body) de la petición. El cliente no debe de usar este método a menos que se cumplan todas las siguientes condiciones:

- La petición contiene la cabecera HTTP *"Content-Type" : "application/x-www-form-urlencoded"*
- El *body* sigue los requerimientos de codificación de la cabecera *Content-Type* definidos por el W3C para HTML 4.01
- El *body* de la petición HTTP es de una sola parte.

- El contenido a ser codificado en el *body* de la petición debe contener únicamente caracteres ASCII
- El método *GET* no puede ser usado.

Un ejemplo del uso de este método está en la siguiente petición HTTP usando seguridad en la capa de transporte(*TLS, Transport-Layer Security*):

POST /recurso HTTP/1.1
Host : servidorRecursos.com
Content-Type : application/x-www-form-urlencoded
access_token = sdj76sdj7iYk

Este método no se debe usar, excepto en el contexto de aplicaciones donde el navegador de los participantes no tenga acceso al campo *Authorization* de la cabecera HTTP de la petición.

3. *URI Query Parameter*

El último método es enviar el *token de acceso* directamente en la URI de la petición HTTP, de la siguiente forma:

GET /recurso?access_token=d658Uih9Uh HTTP/1.1
Host : servidorRecursos.com

Un ejemplo de una URI con un *token de acceso* incluido sería la siguiente:

https:// servidorRecursos.com/recurso?access_token=d658Uih9Uh

Debido a las debilidades de seguridad asociadas con este método, tales como la modificación del token, repetición de tokens o la redirección de la URI destino del

token, ya que existe una alta probabilidad de que la URL que contiene el token de acceso sea accesible por atacantes externos. Por ello, este método sólo debe de usarse en el caso de que sea imposible de enviar el token de acceso mediante alguno de otros dos métodos.

2.11.1.2 OTROS TIPOS DE TOKENS

Aunque la biblioteca descrita en este proyecto sólo considera *bearer tokens*, existen otros tipos. El más importante es el *Mac token*, que utiliza una clave *MAC* junto con el token de acceso. Esta clave se usará para firmar ciertos parámetros de la petición HTTP.

Un ejemplo de una petición que incorporé un token de tipo MAC sería la siguiente:

```
GET /recurso/1 HTTP/1.1
Host : ejemplo.com
Authorization : MAC id = "dfsd796OjLh"
                        nonce="df8TY876shj"
                        mac="dda897UYhjksdsa87hjsSCDS="
```

2.12 USO DE ASERCIONES PAPI COMO AUTHORIZATION GRANT

2.12.1 PAPI

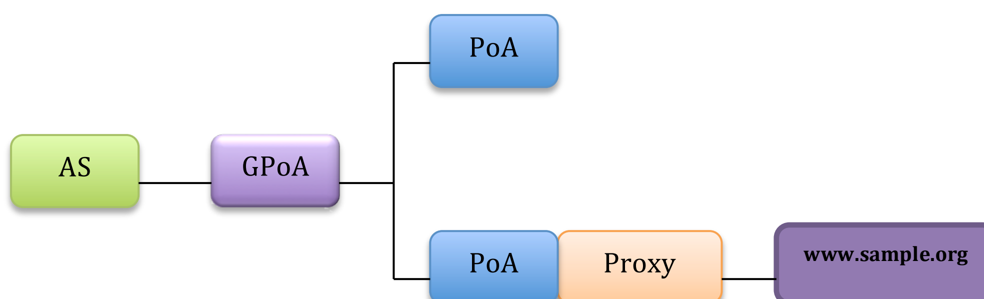
PAPI es un sistema para facilitar el acceso, a través de Internet, a recursos de información cuyo acceso está restringido a usuarios autorizados. Los mecanismos de autenticación empleados para identificar a los usuarios han sido diseñado para ser lo más flexibles posible, permitiendo que cada organización emplee un esquema de autenticación propio, manteniendo así los datos dentro de su propio ámbito, a la vez que los proveedores de información disponen de datos suficientes para realizar estadísticas. Los mecanismos de control de acceso son transparentes para el usuario y compatibles con los navegadores comúnmente empleados en cualquier sistema operativo. Dado que PAPI emplea procedimientos estándar HTTP, su uso para proveer servicios de identidad digital y control de acceso no requiere de ningún hardware o software específico, garantizando a los usuarios un acceso ubicuo a cualquier recurso de información al que tengan derecho.

PAPI es capaz de trabajar en modo federado: Las tareas de identificación y establecimiento de los datos pertenecientes a un usuario (autenticación) se llevan a cabo en el entorno de la organización a la que pertenece, mientras que los procedimientos para establecer derechos de acceso a los recursos (autorización) están bajo el control de la organización que los proporciona. PAPI proporciona los

mecanismos de transferencia de datos y de establecimiento de la confianza entre los participantes necesarios para que este esquema funcione.

El sistema dispone de 4 componentes diferentes:

- **Servidor de autenticación (AS)**: también llamado Proveedor de Identidad se encarga del proceso de autenticar al usuario, validando su identidad y asociándole una serie de atributos para que presente en los recursos protegidos.
- **Punto de Acceso (PoA)**: este componente, conocido como Proveedor de Servicio, realiza la autorización del acceso a un recurso protegido comprobando la autenticación del usuario y sus atributos.
- **Grupo de Puntos de Acceso (GPoA)**: permite centralizar las políticas de autorización de una organización en un sólo punto, al cual preguntarán los Proveedores de Servicio de dicha organización.
- **Proxy PAPI**: es un proxy HTTP con re-escritura de los enlaces con respecto a un recurso web externo. De esta forma, recursos que sólo permiten autenticación por IP pueden integrarse también en una infraestructura de autenticación y autorización.



Las principales características de PAPI son:

- Infraestructura completa para desplegar un sistema de Single Sign-On dentro de una organización o entre diferentes organizaciones, proporcionando la tecnología necesaria para desplegar una federación de identidad digital.
- Proxy web con reescritura de enlaces HTTP, permitiendo el acceso controlado a recursos externos que sólo disponen de mecanismos de autorización básicos como el control por IP.
- Protocolo abierto, ligero y documentado.
- Fácilmente interoperable con otros protocolos de autenticación y autorización, como SAML 1.1, SAML 2, OpenID y OAuth.Software abierto y disponible en diferentes lenguajes de programación: Perl, PHP, Java, ASP.NET, etc.
- Multitud de conectores disponibles para proveedores de servicio: MediaWiki, DokuWiki, Moodle, etc.

2.12.2 ASERCIONES PAPI

Un ejemplo de aserción PAPI, obtenida a través del conector phpPoA(permite conectar aplicaciones con el *Servicio de Identidad de RedIRIS*) es la siguiente:

*“**ePTI**=1a3dfe2027851324541c569e0b9f3fbc1,**ePA**=staff,**sHO**=rediris.es,**ePE**=urn:mace:dir:entitlement:common-lib-*

*terms,**uid**=luis,**sPUC**=urn:mace:terena.org:schac:personalUniqueCode:es:rediris:sir:*

*mbid:{md5}d2f8ff92a3c50a966e007ee56dfd569b***“**

- ***ePTI*** : eduPersonTargetedID, es un identificador persistente, no reasignado ,que preserva la privacidad y que es usado por un *Proveedor de Identidad* en la comunicación con un *Proveedor de Servicios* o un grupo de ellos. Su valor no es revelado a ningún otro *Proveedor de Servicios*, excepto en ocasiones excepcionales.
- ***ePA*** : eduPersonAffiliation, representa la clasificación de afiliación de la persona dentro de la institución en la cual la persona se ha autenticado.
- ***sHO*** : schacHomeOrganization, especifica el nombre del dominio de la institución a la que pertenece la persona que se ha autenticado.
- ***ePE*** : eduPersonEntitlement, especificación del derecho de acceso a, por ejemplo, proveedores de contenidos digitales.
- ***sPUC*** : schacPersonalUniqueCode, identificador único que contiene un hash de la dirección de correo electrónico del usuario.
- ***uid*** : identificador único de usuario requerido para habilitar el uso de OpenID como *Proveedor de Identidad*. El uso de este atributo es opcional.
- ***mail*** : la dirección de correo electrónico del usuario. Es opcional, pero algunos *Proveedores de Servicios* pueden requerir su uso para ofrecer funcionalidades adicionales al usuario o incluso para darles acceso.

En relación al flujo de OAuth2 que realiza la biblioteca implementada en este proyecto, el *Servidor de Autorización* deberá de validar las aserciones PAPI recibidas en relación a las políticas que se hayan establecido en el momento de registro de la aplicación *Cliente* dentro del *Servidor de Autorización*, pudiendo definirse valores específicos para los atributos descritos anteriormente(esto se verá con más detalle en el apartado de manual de usuario).

3. ANÁLISIS DE LA APORTACIÓN REALIZADA

3.1 ANTECEDENTES OAUTH 2.0 EN RedIRIS

Cuando comencé la realización de este proyecto, ya se habían sentado las bases en RedIRIS para la implementación de una biblioteca que implementara el perfil de aserción del protocolo OAuth 2.0. Concretamente estaba implementada la versión 12 de la biblioteca OAuth2lib2_v0.12. Antes de continuar, hay que explicar qué es y qué servicios ofrece el *Servicio de Identidad de RedIRIS, SIR*, para así poder comprender mejor la necesidad de desarrollar una biblioteca de estas características.

3.1.1 SERVICIO DE IDENTIDAD DE RedIRIS, SIR

El **Servicio de Identidad de RedIRIS** (SIR) ofrece un hub de interconexión entre los servicios de identidad de las instituciones afiliadas y proveedores de servicio, a nivel nacional e internacional, ofreciendo a las instituciones que se adhieren al servicio la posibilidad de acceder a recursos protegidos por los

proveedores de servicio que así lo han concertado con RedIRIS.

El SIR se basa en tecnologías de federación de identidades, de manera que:

- Los usuarios se identifican en los servidores locales de la institución, utilizando el procedimiento de identificación definido por la misma y sin que las credenciales sean expuestas fuera del dominio local.
- Los administradores de los servicios de identidad de las instituciones tienen control total sobre los procedimientos de identificación y los atributos asociados a cada usuario.
- Cada institución aplica de manera autónoma los mecanismos de control que considere necesarios para ofrecer a sus usuarios la posibilidad de decidir de manera informada acerca de la información personal susceptible de ser transmitida.
- RedIRIS proporciona una conexión segura, fiable y conforme a estándares entre las instituciones y los proveedores de servicio.
- Los proveedores de servicio aplican de manera autónoma los mecanismos de control de acceso a los recursos que tienen bajo su control. Es importante tener en cuenta que los proveedores de servicio pueden ser tanto entidades ajenas al entorno académico (comerciales, gubernamentales, etc.) como dentro del mismo, ya pertenezcan a RedIRIS u otra red académica nacional (NREN).

La versión actual del SIR utiliza el **protocolo de federación PAPI v.1** y soporta los

siguientes protocolos de salida:

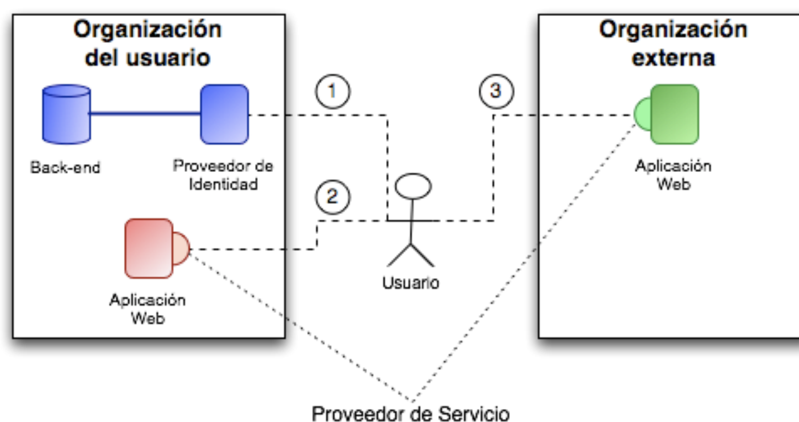
- PAPI v.1
- SAML 1.1 / Shibboleth 1.3
- SAML 2 / Interoperable SAML2 Profile / Shibboleth 2
- eduGAIN con perfil SAML 2.0 Int
- OpenID (versión 1 y 2)
- Protocolos propietarios:
- Microsoft Live@Edu SSO
- MSDN Academic Alliance
- Wiley Trusted Proxy Server

El servicio se basa en la publicación de los datos de confianza a través de los metadatos.

La lista de instituciones que se han unido al servicio es muy extensa, sobre todo de carácter académico(numerosas universidades españolas, bibliotecas públicas), institucional(el Senado, Boletín Oficial del Estado) o científico(CICA, Donostia International Physics Center) .

También existen una serie de proveedores de servicios que, potencialmente, pueden ser utilizados en esta federación. Estos son, por ejemplo *Google Apps for Education*, *Microsoft Live@Edu*.

Las **infraestructuras federadas de autenticación y autorización** tienen como objetivo delegar la identificación de cada usuario en la organización a la que pertenece, en la cual se realizará el proceso de autenticación una sola vez. Este proceso es conocido como **Single Sign-On**. Cada vez que el usuario quiera acceder a un recurso protegido, tendrá que presentar las credenciales obtenidas al autenticarse.



De esta forma, podemos ver que cuando implantamos una federación, el usuario quiere acceder a determinados recursos protegidos tanto en su organización como en una externa. En este nuevo escenario identificamos dos componentes principales en una federación:

- **Proveedor de identidad**: realiza la autenticación del usuario y emite sus credenciales. éstas no sólo contienen información sobre si la autenticación ha sido correcta, sino que también incluirá una serie de atributos asociados

a él. En el caso de una universidad, reflejaría por ejemplo si el usuario es un alumno o un profesor. El objetivo es reducir al máximo la información más sensible del usuario, como su nombre y apellidos, y basar los derechos de acceso en qué tipo de usuario es.

- **Proveedor de servicio**: comprueba las credenciales del usuario, y en el caso de que no sean válidas o suficientes, o bien el usuario no las haya obtenido previamente, deniega el acceso al recurso protegido.

Para acceder a un recurso protegido en una federación, en un primer momento (*paso 1 en la figura anterior*) realizará el proceso de autenticación en el proveedor de identidad de su organización. Gracias a esta acción, obtendrá sus credenciales, las cuales generalmente están cifradas de manera que sólo el proveedor de identidad y los proveedores de servicio puedan leerlas. En el momento que quiera acceder a una aplicación web de su organización (*paso 2 en la figura anterior*), no tendrá que volver a identificarse, sino que en la misma petición envía también sus credenciales. Nos encontramos con el mismo caso cuando acceda a la aplicación web (*paso 3 en la figura anterior*) de la organización externa que ha instalado un proveedor de servicio para proteger su recurso.

Las ventajas que ofrece este escenario federado son:

- Las organizaciones no tienen que registrar los datos de los usuarios de otras organizaciones. Además de no aumentar los costes de tiempo en gestionar usuarios de otras entidades, se reduce la complejidad de las

políticas de seguridad y no interfiere en el cumplimiento de las leyes de protección de datos.

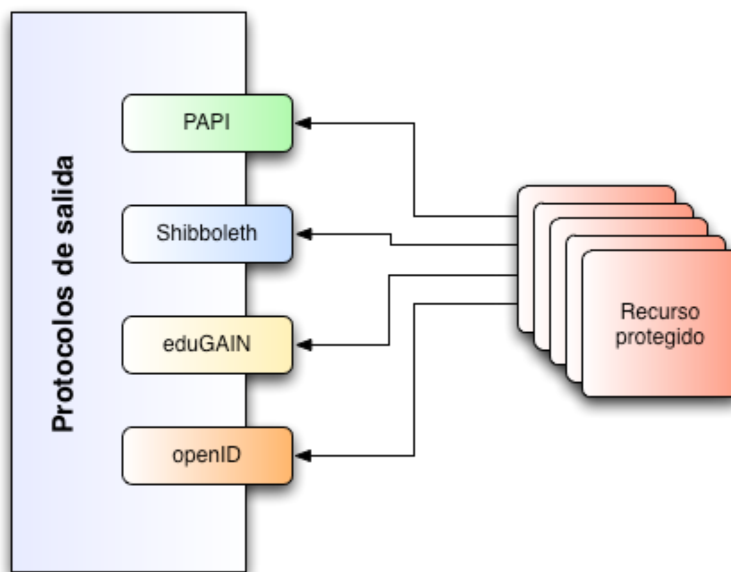
- Los usuarios sólo tienen que recordar su nombre en el sistema y su contraseña. Aumenta la seguridad de la infraestructura.

Los recursos son ofrecidos a los usuarios de todas las organizaciones que participan en la federación, sin que el usuario tenga que solicitar el acceso a cada uno de los recursos.

3.1.1.1 CARACTERÍSTICAS DEL SERVICIO DE IDENTIDAD DE RedIRIS

Una vez que la institución se incluye en el servicio se convierte un proveedor de identidad válido dentro de la federación gestionada por el SIR.

De esta forma, se le facilita a todo proveedor de identidad una serie de interfaces de salida que pueden ser utilizadas por las instituciones, incluso fuera del ámbito de este servicio:



- **PAPI :** La interconexión de una institución en esta federación se basa en el protocolo definido por PAPI v1, por lo cual todo proveedor de identidad tiene disponible su propio **Servidor de Autenticación** (AS) de PAPI, una vez que ha terminado de unirse a este servicio.
- **Shibboleth :** Cada proveedor de identidad también tiene asociado un *Identity Provider* (IdP) de Shibboleth, permitiendo así poder acceder recursos protegidos por dicha tecnología, como por ejemplo Microsoft DreamSpark. El IdP se anuncia en un fichero de metadatos, base de la confianza de la federación, donde se puede obtener información sobre su despliegue. En el caso de que la institución ya tuviera desplegado un IdP, RedIRIS ofrece la posibilidad de incorporar su metadato en el SIR por lo cual no sería necesario que tuviera otro IdP en el SIR.
- **eduGAIN :** SIR es el punto de salida desde las instituciones afiliadas a

RedIRIS hacia eduGAIN, una red de federaciones europeas promovida por la red académica paneuropea GéANT2. En el momento que haya recursos protegidos a través de este servicio, estarán disponibles automáticamente en el SIR.

- **openID :** Todos los usuarios de los proveedores de identidad en SIR tienen un identificador de OpenID (tanto en su protocolo v1 como en v2). El nombre de usuario viene dado de la siguiente forma: ***http://yo.rediris.es/soy/nombre_de_usuario@dominio_institucion.org*** donde:

  - *nombre_de_usuario*: es el nombre del usuario proporcionado por su institución
  - *dominio_institucion.org*: es el dominio principal en internet de la institución del usuario

En caso de duda, existe un servicio, PAPOID Finder, en SIR que informa al usuario de su identificador de openID, evitando la molesta de preguntar en su proveedor de identidad.

Además, existen versiones del identificador también en catalán, euskera y gallego, disponibles para todos los usuarios.

3.1.2 INTEGRACIÓN DEL PROTOCOLO OAUTH CON EL SIR

Este proyecto tenía como objetivo explorar la posibilidad de integrar el

Servicio de Identidad de RedIRIS con el protocolo OAuth, al igual que interacciona actualmente con otros protocolos como OpenID o PAPI.

OAuth permite a una aplicación cliente acceder y procesar los recursos de un usuario en su nombre, siempre y cuando el propietario del recurso haya dado autorización para ello.

La idea por la cual se veía la necesidad de realizar este estudio era que OAuth ofrece ciertas ventajas de las que el SIR puede beneficiarse, como la flexibilidad del protocolo en cuanto a configuración, la facilidad de uso y de implantación por parte de un cliente y elementos de seguridad que establece, ya que evita que la aplicación cliente tenga credenciales de los usuarios. Además permite incorporar aplicaciones personalizadas a los servicios ofrecidos por las instituciones.

Como primer caso de uso para explorar este funcionamiento, se decidió implementar una aplicación cliente que permitiera acceder de forma segura a la información relativa a la inscripción y registro de las listas de distribución de RedIRIS.

Como solución a este caso de uso se realizó la implementación en código PHP de una **aplicación cliente** que tratara la información de las listas de correo, pero haciéndola genérica para que cualquier institución pudiera desplegar estas de forma transparente, sencilla, haciendo su uso casi transparente a la persona encargada de su despliegue. Además se codificó una **aplicación servidora de recursos**, que tratara las peticiones de las aplicaciones clientes, realizase la autorización y autenticación del usuario a los recursos y administrara el acceso a

los recursos que requería la aplicación cliente. Además fue necesario implementar una **aplicación de registro** de las aplicaciones clientes para que el servidor de los recursos pudiera conocer qué aplicaciones son dignas de confianza.

Este caso de uso inicial se realizó en la versión 1.0a del protocolo OAuth, pero al ver que ésta se quedaba obsoleta y que nuevas versiones y extensiones del protocolo ofrecían más posibilidades al SIR, se decidió realizar una biblioteca en PHP de la nueva versión de oauth, OAuth2, la cual hemos denominado OAuth2lib.

La segunda versión del protocolo OAuth se encuentra actualmente en *draft* en la organización estandarizadora *IETF* y nos ofrece una modificación importante a la versión inicial del protocolo, ya que permite que la autorización para acceder a los recursos no tenga que realizarse directamente con el usuario, si no que se puede delegar en un Servidor de Autorización. Este procedimiento nos permite utilizar aserciones tales como PAPI o SAML, por lo que la perspectiva de integración con el SIR se ve ampliada exponencialmente.

3.1.3 BIBLIOTECA OAUTH2 PARA EL PERFIL DE ASERCIÓN, RedIRIS

OAuth2lib es una biblioteca PHP basada en el perfil de aserción de OAuth 2.0 diseñada con la intención de dar soporte a la integración del protocolo OAuth con el *Servicio de Identidad de RedIRIS.*

En esta biblioteca se ha implementado el flujo de autorización llamado *Perfil de Aserción*, del cual se habló en el capítulo 2, y está diseñada para usar tanto

aserciones PAPI como aserciones SAML2 como *authorization grant*. El detalle del diseño e implementación de esta biblioteca se verá en el próximo capítulo, ya que se trata del punto de partida de la biblioteca implementada en este proyecto.

3.1.4 ENTORNO SIR PARA EL PROTOCOLO OAUTH2

El entorno de OAuth2 dentro del SIR, *Sirope*, pretende ofrecer a los usuarios de entidades que pertenezcan al SIR servicios que puedan serles útiles.

Actualmente se ofrece una forma de conocer los proveedores de servicios a los cuales un usuario puede acceder.

Además de ofrecer esta aplicación cliente, RedIRIS ofrece un **Servidor de Autorización** y un **Servidor de Recursos** que funcionan con el protocolo OAuth2, haciendo posible que cada entidad pueda tener su propia Aplicación Cliente de OAuth2 y que ésta se comunique con los servidores de RedIRIS. El código de este entorno forma parte del proyecto OAuth2lib, el cual se puede consultar en la *Forja* de RedIRIS

3.2 OBJETIVOS PARA LA MEJORA DE LA BIBLIOTECA

Con la finalidad de realizar mejoras en la biblioteca ya implementada, se definieron una serie de objetivos que se detallan a continuación.

3.2.1 FORMATO DE TOKENS JWT(JSON Web Token)

Primero vamos a estudiar qué es un token JWT y como es su estructura. JSON Web Token, JWT, es un formato de tokens diseñado para entornos con limitaciones de espacio, tales como las cabeceras HTTP de autorización (HTTP Authorization Headers) o los parámetros URI que van en las consultas. JWT codifica reclamos o reivindicaciones(a partir de ahora serán llamados ***claims***) para que sean transmitidos usando la notación de objetos JavaScript, JSON. Estos objetos serán codificados usando base64url(codificación en base64, descrita en [RFC4648], con el carácter de padding '=' omitido) y deben de estar firmados y/o encriptados. La firma se realiza usando *JSON Web Signature*, JWS(draft-ietf-jose-json-web-signature-04); el cifrado se realiza usando JSON Web Encryption, JWE(draft-ietf-jose-json-web-encryption).

El conjunto de ***claims*** JWT representa al objeto JSON, consistente en cero o más pares de nombre/valor, donde los nombres son cadenas de caracteres y los valores son valores arbitrarios JSON. Estos miembros son los ***claims*** representados por el JWT.

Un token JWT está formado por las siguientes partes(cada una va codificada en base64url por separado):

- ***JWT header***, un String que representa un objeto JSON que describe las operaciones criptográficas aplicadas al JWT("*alg*"), así como el tipo de token JWT("*typ*").

- ***JWT claims set***, un String que representa un objeto JSON que contiene los claims transmitidos por el JWT.
- ***Firma JWT***, se trata de la firma digital de la concatenación del ***JWT header*** con el ***JWT claims set***, usando como método criptográfico de firma el descrito en la cabecera "*alg*" del JWT Header.

Un ejemplo de un token JWT podría ser el siguiente(cada parte se ha codificado en base64url por separado):

JWT Header *: {"typ":"JWT",*

"alg":"HS256"}

JWT claims *: {"iss":"joe",*

"exp":1300819380,

"http://example.com/is_root":true}

Firma JWT *: dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk*

JWT : *"eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9*

.

eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGF tcGxlLmNvbS9pc19yb290Ijp0cnVlfQ

.

dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk"

El soporte para que el *Servidor de Autorización* pueda generar este tipo de *tokens de acceso* era uno de los objetivos de este proyecto. Para ello se crearon dos clases dentro de la biblioteca oauth2lib. Una estaría dentro del *Servidor de Autorización* y establece la lógica de creación de *tokens JWT*. La otra forma parte del *Servidor de Recursos* y tiene como fin decodificar los *tokens JWT* recibidos en la petición de un *recurso protegido.*

A continuación se muestra un ejemplo un *token de acceso* JWT, codificado y en texto plano, generado por la biblioteca de este proyecto:

TOKEN

“*eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzEiLCJwYXYiOiIyLjAifQ.eyJpYXQiOjEzNDI5NDQ1OT*
MuNjMzNiwiZXhwIjoxMzQyOTQ4MTkzLjYzMzYsInR5cCI6Ik9BdXRoSldUIiwiaXNzIjoi
YXV0aHNlcnZlcl9leGFtcGxlIiwiY2xpZW50X2lkIjoicHJ1ZWJhX29hdXRoX1NBTUwiLCJ
0b2tlbl9pbmZvIjp7InVybjptYWNlOnRlcmVuYS5vcmc6c2NoYWM6YXR0cmlidXRlLWR
lZjpzY2hhY0hvbWVPcmdhbml6YXRpb24iOiJyZWRpcmlzLmVzIiwidXJuOm1hY2U6ZGl
yOmF0dHJpYnV0ZS1kZWY6bWFpbCI6Imx1aXMuZ29tZXpAcmVkaXJpcy5lcyJ9LCJzY2
9wZSI6InNjb3BlX2RlX3BydWViYV9TQU1MIiwiYXV0aHpJRCI6IjIwODFkNDZmODIyO
DMyNjliMjc3NWMwOWFkYzAyZjIxOWZjNGM5ZjRjNTlkMmYxZTliOGZhZmNmZmYx
MDJhYTcifQ.3gFgfbZJVyc1PbHvckD8Zsr8SGmKEZ3jUen-UDfg11k5l0-
13UNTaBiHofXWYUtR1guImEzjWeXkN99UJHBsqCW8mpwH3hZRVkfsPBxC_PYqItkY
Z1dh1P5uL8bke35qMJLf3jI1iNmC4gYYPgMN3p5OZrAYmFndFPz0GT6Mvoc”

TOKEN DECODIFICADO

Array

(

[typ] => OAuthJWT

[iss] => authserver_example

[iat] => 1342944593.6336

[exp] => 1342948193.6336

[client_id] => prueba_oauth_SAML

[token_info] => Array

(

[urn:mace:terena.org:schac:attribute-def:schacHomeOrganization] => rediris.es

[urn:mace:dir:attribute-def:mail] => luis.gomez@rediris.es

)

[scope] => scope_de_prueba_SAML

[authzID] => 2081d46f82283269b2775c09adc02f219fc4c9f4c59d2f1e9b8fafcfff102aa7

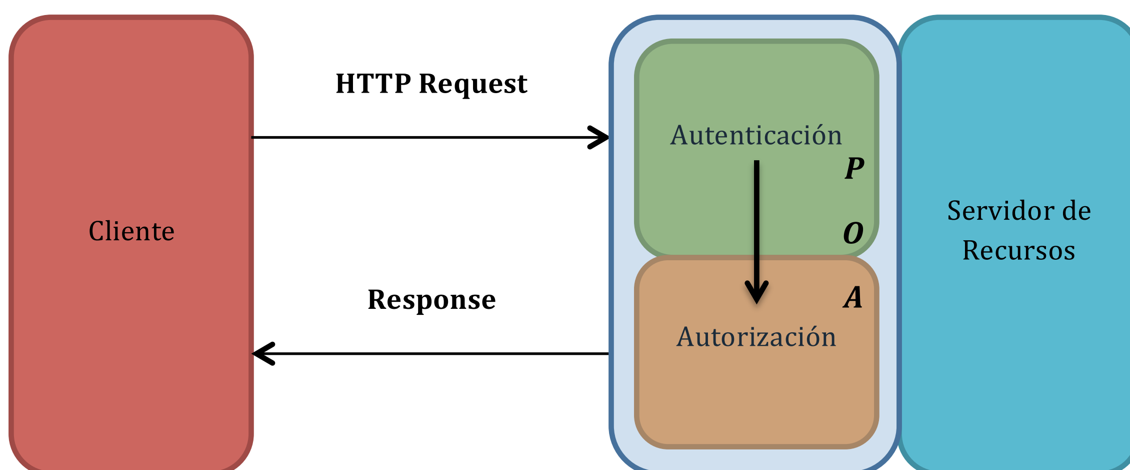
)

3.2.2 AUTENTICACIÓN Y AUTORIZACIÓN phpPoA EN SERVIDOR DE RECURSOS

Para este objetivo, la idea era añadir el flujo de autenticación y autorización de un *token de acceso* dentro de la funcionalidad de phpPoA. Recordemos que este proceso, de verificación y validación de un *token de acceso* es llevado a cabo por el *Servidor de Recursos*. Para ello, el *Servidor de Recursos* debe de ser capaz de poder hacer una llamada a los métodos de la biblioteca phpPoA ***authenticate*** e ***isAuthorized***, usando para ello dos nuevos motores de autenticación y autorización diseñados en el desarrollo de este proyecto.

A continuación se muestra un figura con el flujo que sigue una petición de un recurso protegido por parte de un *Cliente* a un *Servidor de Recursos* haciendo uso

para ello del proceso de autenticación y autorización que se puede desplegar sobre phpPoA:



En la petición HTTP el cliente envía el *token de acceso recibido por un* Servidor de Autorización. El PoA puede rechazar la petición si no es capaz de validarlo o si este no está autorizado para el alcance del recurso que está solicitando. En alguno de los casos anteriores, se debe de devolver un mensaje de error OAuth al cliente explicando los motivos del error.

3.2.3 REPASAR CLIENTE

Durante el estudio y elaboración de la biblioteca de este proyecto se encontraron dos problemas en la implementación de la aplicación cliente. Estos son:

- Incapacidad de una aplicación cliente de separar la lógica de petición de *tokens de acceso* con la de petición de *recursos protegidos*. Es decir, en versiones anteriores de la biblioteca, el flujo de acceso a recursos protegidos se hacía en una única función, la cual se encargaba de realizar la petición de un *token de acceso* a un *Servidor de Recursos,* presentando una aserción válida, para inmediatamente solicitar el *recurso* al *Servidor de Recursos* presentando el *token de acceso* recibido. Realmente, realizar las dos peticiones dentro de la misma función no tiene mucho sentido(ya que, por ejemplo, el tiempo de vida de un token no serviría para mucho, ya que se usa inmediatamente se ha recibido).
- Relacionado con lo anterior, si el token no tiene por qué ser usado inmediatamente una vez se ha recibido, se debe de permitir que este *token de acceso* pueda ser almacenado temporalmente dentro de la aplicación cliente. Es por esto la necesidad de añadir a la biblioteca métodos que permitan al cliente almacenar temporalmente *tokens de acceso*. Con este propósito, se han implementado tres métodos para el almacenamiento de tokens:
    - ***session***, guarda el *token de acceso* en una variable de sesión PHP($_SESSION), formando una estructura que incluye el identificador del *propietario del recurso*, el *token de acceso* almacenado, y el tiempo de expiración del mismo

- ***db,*** almacena el token en una base de datos simple de tipo dba. La estructura almacenada es idéntica a la anterior, sólo varía el lugar de almacenamiento.
- ***file,*** guarda el *token de acceso* en una variable PHP dentro de un fichero con extensión PHP(es el método menos recomendado de los tres, debido a que ha de pulirse mejor su comportamiento).

3.2.4 OTROS OBJETIVOS

A parte de estos objetivos, durante el desarrollo del proyecto se tuvieron que corregir algunos fallos encontrados en la implementación de la biblioteca

- ***AÑADIDO SOPORTE PARA ASERCIONES SAML2*** : durante la realización del primer objetivo, me di cuenta de que el *Servidor de Autorización* no realizaba ningún tipo de comprobación sobre la validez de las aserciones SAML2(los descritos en el punto *2.7.2.2 FORMATO DE LA ASERCIÓN Y REQUISITOS DE PROCESAMIENTO*). Es por ello que este punto paso a ser uno de los objetivos principales de este proyecto, dar soporte real al perfil de aserciones SAML2 especificado por OAuth2.
- ***MEJORA EN LA IMPLEMENTACIÓN DE LAS POLÍTICAS*** : este fue el otro problema digno de mención que me encontré en la realización de este

proyecto. Una política es una serie de comprobaciones, que realiza el S*ervidor de Autorización,* sobre los atributos de las aserciones que le llegan en las peticiones de *tokens de acceso*. Las políticas son almacenadas en el fichero de configuración referente a las políticas, y están distribuidas por *scope*. El problema era que la biblioteca no era capaz de comprobar más de una política dentro de un mismo *scope*, y esto presentaba un problema ya que la biblioteca debería de dar soporte para más de una política por scope. El cómo funcionan los archivos de configuración de las políticas se explicará más adelante, en el capítulo de Manual de Usuario.

4. ANÁLISIS DE REQUISITOS, DISEÑO E IMPLEMENTACIÓN

Los requisitos de esta biblioteca son los descritos en apartados anteriores, necesarios para implementar el protocolo OAuth2 en su perfil de aserción y conectarlo con el *Servicio de Identidad de RedIRIS*. Los detalles del diseño y la implementación de la biblioteca se pasan a describir a continuación, y básicamente están divididos en tres partes: *Cliente*, *Servidor de Autorización* y *Servidor de Recursos*.

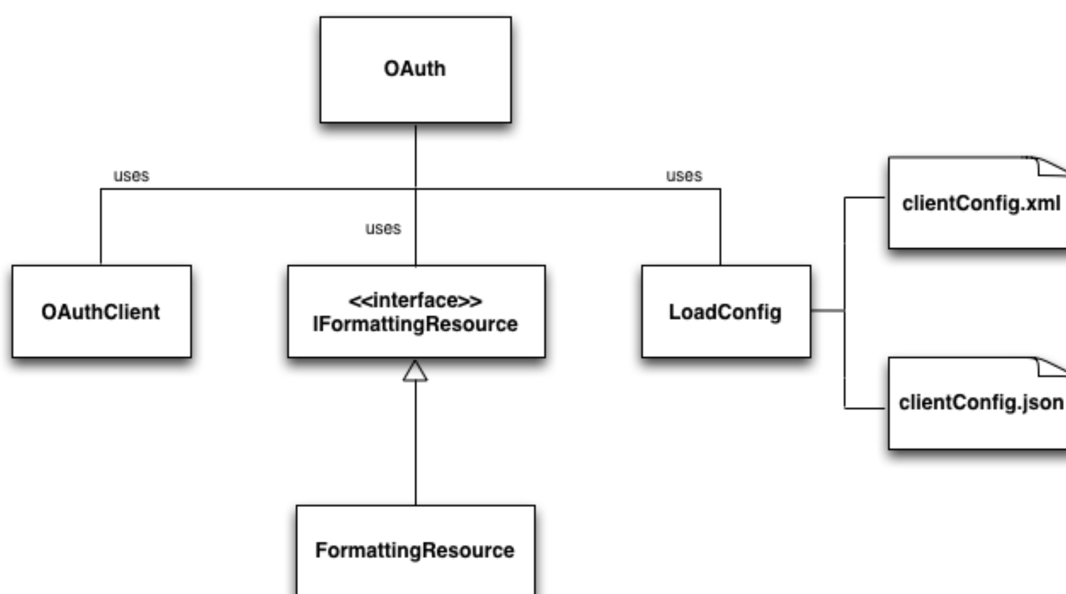
4.1 CLIENTE

La aplicación cliente tiene los siguientes objetivos:

1. Hacer la petición de un *token de acceso* al *Servidor de Autorización*, presentando para ello una aserción.

2. Hacer la petición del recurso en nombre del propietario presentando el *token de acceso* recibido.

La siguiente figura muestra el diagrama de clases usado en el diseño de la arquitectura del *Cliente*:



El contenido de la aplicación cliente se encuentra en la carpeta *oauth_client* dentro del proyecto oauth2lib. A continuación se pasa a describir el contenido de la carpeta de la aplicación cliente:

- ***client_token*** : carpeta que contendrá los archivos necesarios para el almacenamiento temporal de los *tokens de acceso* de los usuarios de la aplicación cliente(sólo útil si los métodos de almacenamiento son *dba* o *file*).

- ***config*** : carpeta que contiene los archivos de configuración que controlan el comportamiento de la aplicación cliente. En el lado *Cliente* es posible establecer esta configuración en dos formatos distintos: *JSON* o *XML*.
- ***src*** : carpeta que contiene la clase principal OAuth, encargada de realizar el flujo de autorización y petición de recursos protegidos. Para la autorización(método ***requestAccessToken***), dependiendo del método de almacenamiento establecido para el cliente, comprueba si existe un token almacenado para el usuario y, en caso de existir, comprueba si sigue siendo válido. Si no existe token, realiza una petición al *Servidor de Autorización* presentando las credenciales establecidas para el perfil Aserción de OAuth2. Una vez obtenido el token, se puede hacer uso del método ***requestResourceRS.*** Para ello hace uso de las clases e interfaces siguientes:
    - ***OAuthClient*** , se establece como tipo de un atributo de la clase OAuth, *oauth_client*, que se usará para llamar al método de clase ***doAccessTokenRequest***, el cual realiza la petición cURL al *Servidor de Autorización*. También contiene al método ***requestResource*** para realizar la petición del recurso al *Servidor de Recursos* una vez obtenido el *token de acceso*.
    - ***LoadConfig o LoadConfigJSON***, se encarga de cargar y almacenar en un atributo de la clase OAuth el archivo de configuración del cliente. Dependiendo de si la configuración está en *XML*(*LoadConfig*) o en formato *JSON*(*LoadConfigJSON*).

- ***IFormattingResource***, es la interfaz que contiene los métodos que darán formato al recurso obtenido. Debe de estar implementada en una o más clases, que darán forma al recurso dependiendo del tipo que sea(cadena de caracteres, archivo en cualquier formato, estructura JSON). En la biblioteca entregada junto a este proyecto, existe una clase llamada FormattingResource que muestra un ejemplo bastante sencillo de cómo se relaizaría la tarea de dar formato al recurso.

A continuación se muestra un ejemplo de los parámetros enviados por el *Cliente* implementado en la biblioteca de este proyecto cuando realiza una petición de un *token de acceso*:

HTTP Header : "***Authorization : Basic ddFe45dfE45fDSeds3***"

Post fields : "***grant_type*** = *assertion*

assertion_type = *urn:mace:rediris.es:papi*

assertion = *ePTI=...,ePA=f, sHO=rediris.es =...*

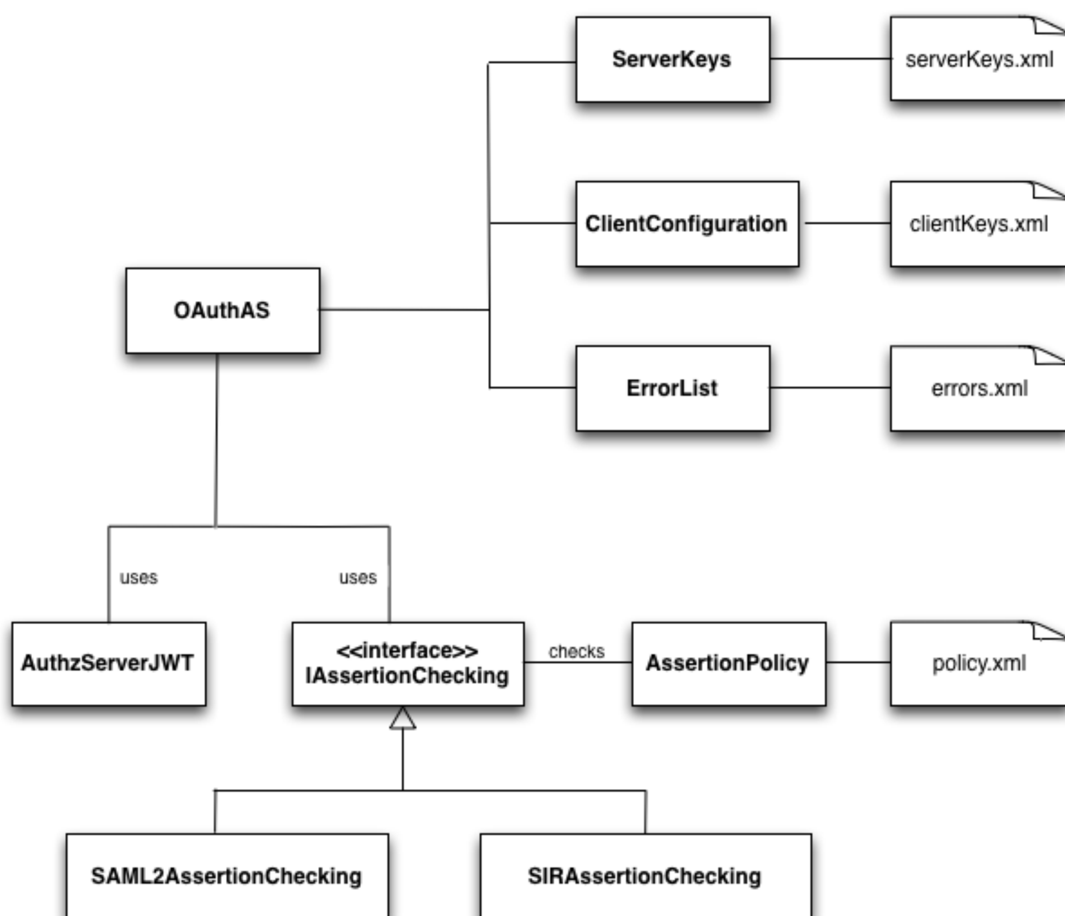
scope = *scope_de_prueba*

client_id = *cliente_de_prueba*"

4.2 SERVIDOR DE AUTORIZACIÓN(AS)

El objetivo del *Servidor de Autorización* es, dada una aserción, comprobar que es válida y generar un *token de acceso* para un *scope* específico.

A continuación se muestra un diagrama de clases del *Servidor de Autorización* incluido en la biblioteca de este proyecto:



El contenido del *Servidor de Autorización* se encuentra en la carpeta *oauth_as* del proyecto oauth2lib. Su contenido es el siguiente:

- ***config*** : carpeta que contiene los archivos de configuración que controlan el funcionamiento del *Servidor de Autorización*. Estos son:
    - ***clientKeys.xml*** : contiene las credenciales y la configuración de las aplicaciones *Cliente* registradas en el *Servidor de Autorización*.
    - ***errors.xml*** : contiene un identificador y una descripción de todos los posibles errores que puede devolver el Servidor de Autorización a un *Cliente* en la petición de un *token de acceso*.
    - ***policies.xml :*** Contiene las políticas que debe de seguir el *Servidor de Autorización* a la hora de validar las aserciones recibidas. Se divide en configuración para aserciones PAPI y para aserciones SAML, definiendo dentro de cada una las políticas para cada *scope*.
    - ***serverKeys.xml*** : contiene las credenciales de todos los *Servidores de Recursos* que son capaces de validar tokens emitidos por el *Servidor de Autorización* que se está configurando.
- ***src*** : carpeta que contiene las siguientes clases e interfaces:
    - La clase principal ***OAuthAS***, encargada de gestionar todo el proceso de manejo de peticiones de *tokens de acceso*. Para ello, valida, por este orden, los siguientes elementos de la petición:
        - Comprueba que la petición tiene un formato válido(método ***isValidFormatRequest***).

  - Comprueba que el *scope* recibido está registrado como válido para el *Cliente* del que proviene la petición(***isValidScope***).
  - Comprueba que la petición viene de un cliente registrado(**isValidClient**), para ello, valida la cabecera HTTP

    Authorization: Basic <VALOR CIFRADO>

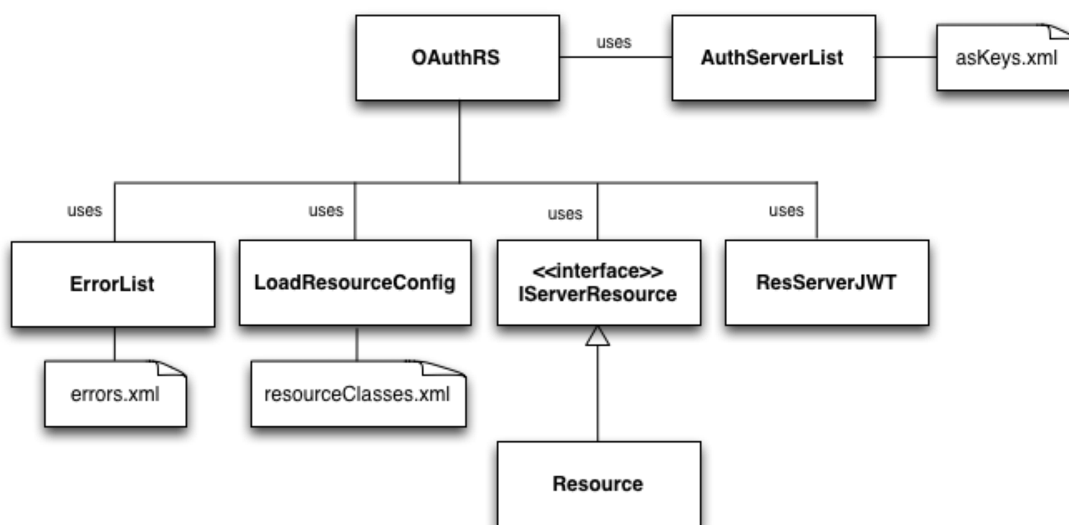
    siendo <VALOR CIFRADO> el valor del *client_id* cifrado mediante HMAC(sha256), usando como clave el *cliente_secret*.
  - Por último comprueba la validez de la aserción(**isValidAssertion**), generando un *token de acceso* por el método(directamente en el *Servidor de Autorización* o a través de un STS, Security Token Service) que tenga configurado el *cliente* del que provenga la petición.
- La carpeta ***assertions***, que contiene a su vez:
  - La interfaz ***IAssertionChecking***, que define los métodos para cualquier clase de comprobación de aserciones. Se ha diseñado de tal forma que para añadir soporte a un nuevo tipo de aserciones, habría que crear una clase que implemente esta interfaz y que de soporte al nuevo tipo de aserción.
  - Las clases ***SAML2AssertionChecking*** y ***PAPIAssertionChecking***, encargadas de comprobar la si la aserción, *SAML2 o PAPI* respectivamente, cumple con el formato y la política adecuados.

        - La clase ***AssertionPolicy***, que maneja el archivo de políticas ***policies.xml***, que contiene las políticas de autorización para cada tipo de aserción.
    - La clase ***AuthzServerJWT***, que incluye la lógica de creación de *tokens de acceso* de tipo JWT. El diseño y la estructura de este tipo de *tokens de acceso* es el explicado en el capítulo anterior en el apartado de objetivos.
- ***tokenEndpoint***, es el punto de acceso al *Servidor de Autorización*, encargado de recibir la petición de un *token de acceso*, extraer los parámetros de la petición y construir con ellos un objeto de la clase ***OAuthAS*** que procese la petición.

4.3 SERVIDOR DE RECURSOS(RS)

El objetivo del S*ervidor de Recursos* es, dado un *token de acceso*, comprobar si este es válido y, en caso afirmativo devolver el recurso solicitado. Si el *token de acceso* está expirado, no es válido o presenta algún otro problema la petición, el *Servidor de Recursos* devuelve un mensaje de error al *Cliente*.

La siguiente figura muestra el diagrama de clases usado en el diseño de la arquitectura del *Servidor de Recursos*:



El contenido del *Servidor de Recursos* se encuentra en la carpeta *oauth_server* del proyecto oauth2lib. Su contenido es el siguiente:

- Una carpeta ***config*** que contiene los ficheros de configuración necesarios para que funcione el *Servidor de Recursos*:
  - ***asKeys.xml***, contiene un identificador, una URL y una contraseña por cada *Servidor de Autorización* del cual el *Servidor de Recursos* es capaz de validar *tokens de acceso*.
  - ***errors.xml***, contiene un identificador y una descripción de todos los posibles errores que puede devolver el Servidor de Recursos a un *Cliente* en la petición de un *recurso protegido*.
  - **resourceClasses.xml**, especifica los atributos(PAPI o SAML2) que deben de estar contenidos en el *token de acceso*, así como las clases

que se usarán para dar formato al recurso. Esta información está almacenada en el archivo de configuración distribuida por *scopes*.

- Una carpeta ***src*** que contiene las siguientes clases e interfaces:
    - ***OAuthRS***, clase encargada de realizar toda la lógica del proceso de validación de *tokens de acceso* y envío de *recursos protegidos*. Para ello, primero comprueba la validez la petición(**isValidFormatRequest**), para luego comprobar la validez del *token de acceso* recibido(**isValidToken**). Si todo es correcto, pasa a enviar el *recurso* al *Cliente* que lo solicitó. En caso contrario, devuelve un mensaje de error.
    - ***ResServerJWT***, clase que incluye la lógica para la validación de *tokens de acceso* de tipo *JWT*.
    - Clases para cargar los archivos de configuración. Estas son ***AuthServerList***, ***ErrorList*** y ***LoadResourceConfig.***
    - La carpeta ***resources***, que incluye la interfaz ***IServerResource*** que define los métodos para dar formato al *recurso*. Junto a esta interfaz, hay una clase de prueba que sirve como ejemplo de implementación de la interfaz anterior.
- ***serverEndpoint***, es el punto de acceso al *Servidor de Recursos*, encargado de recibir la petición de un *recurso*, extraer los parámetros de la petición y construir con ellos un objeto de la clase ***OAuthRS*** que procese la petición.

5. ANÁLISIS TEMPORAL Y DE COSTES

5.1 ESTIMACIONES INICIALES

Al comenzar el proyecto, se estableció una temporización aproximada, basándonos en los datos que teníamos inicialmente:

- El proyecto fin de carrera debía ocupar 540 horas como mínimo.
- El inicio del proyecto fue en Julio de 2011.
- Desde esa fecha, el tiempo diario dedicado sería de 5 horas, de lunes a viernes.

Con esta información pasamos a dividir en partes el trabajo necesario para realizar el proyecto: Análisis de los requisitos, Documentación, Implementación y Realización de la presentación.

Análisis de los requisitos: Comprenderá las actividades de planificación, organización, investigación de tecnologías y herramientas, reuniones iniciales y de *brainstorming.*

Documentación: Actividades de desarrollo de la documentación escrita, manuales de usuarios, guía de instalación, gráficos y tablas explicativas, etc.

Implementación: Todo lo relacionado con la implementación de la solución definida.

Presentación: Comprende el esfuerzo dedicado a la realización de la presentación, preparación de la documentación que tuviera que entregarse, grabación de CDs con el código, etc.

En este año de trabajo, se llegaron a estimar los siguientes valores para cada una de las actividades:

- ***Análisis***: Ocuparía la primera fase del proyecto, siendo el esfuerzo de mes y medio aproximadamente.
- ***Implementación***: Sería la segunda fase del proyecto, con una duración de dos meses aproximadamente.
- ***Documentación***: Tercera fase del proyecto, de mes y medio aproximadamente.
- ***Presentación***: Última fase, con un esfuerzo de medio mes.

Debido a la realización de anteriores proyectos fin de carrera, sabíamos que estas etapas, aunque iban a estar más o menos repartidas en el tiempo, acabarían por solaparse, por lo que no pusimos fechas muy exactas en la temporización de cada una de las etapas.

5.1 ESTIMACIONES FINALES

Durante la realización del proyecto, se fue completando una ficha con las horas dedicadas para cada etapa, para así poder comparar los valores estimados con los reales.

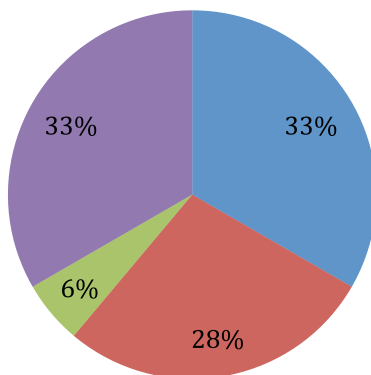
Las estimaciones finales obtenidas se han esquematizado en la siguiente tabla:

ETAPA	TIEMPO ESTIMADO	TIEMPO INVERTIDO
Análisis	180	225
Documentación	150	180
Implementación	180	170
Presentación	30	30
Horas totales	540	605

Las siguientes gráficas muestran una comparativa entre el *Tiempo Estimado* y el *Tiempo Invertido*

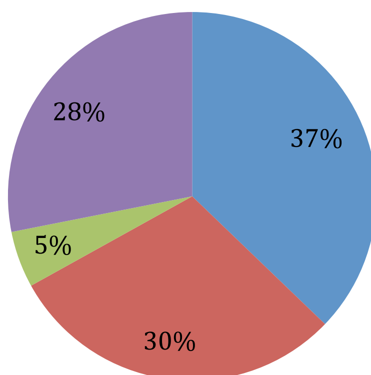
Tiempo Estimado

Análisis Documentación Presentación Implementación



Tiempo Invertido

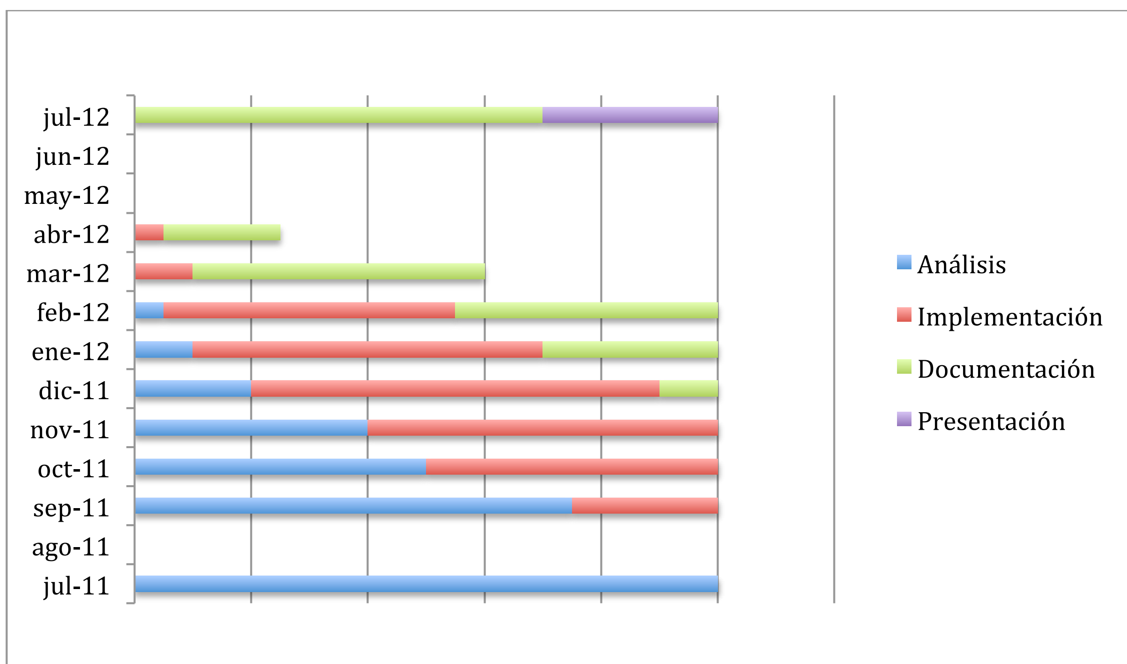
Análisis Documentación Presentación Implementación



Como se puede observar en las gráficas y la tabla, se estimó una duración menor para todas las etapas, salvo para la etapa de *Implementación*. La etapa que más diferencia presenta entre lo estimado y lo invertido es la etapa de Análisis, y esto es debido a la dificultad que presentó el estudio del protocolo OAuth2, así como de las bibliotecas ya implementadas por RedIRIS. Este tiempo también se vio aumentado influido por la necesidad de corregir los problemas no previstos de antemano, tales como el no soporte para aserciones SAML2 o la implementación con fallos de las políticas.

También cabe destacar que se estimó un tiempo de implementación mayor del que requirió finalmente. Éste se vio reducido gracias a que aplicamos más esfuerzo al análisis y diseño de la solución, por lo que la implementación fue una tarea de aplicación directa.

Por último, en este análisis de costes temporales, mostramos la gráfica de ocupación de cada etapa dependiendo del momento en el que se realizaron. Como podemos ver, aunque cada etapa está restringida a un momento determinado del desarrollo, se solapan muchas de ellas en el tiempo, tal y como habíamos previsto en las estimaciones iniciales.


jul-12
jun-12
may-12
abr-12
mar-12
feb-12
ene-12
dic-11
nov-11
oct-11
sep-11
ago-11
jul-11
Análisis
Implementación
Documentación
Presentación

6. MANUAL DE USUARIO

6.1 ARCHIVOS DE CONFIGURACIÓN

A continuación se muestran los archivos de configuración que controlan el comportamiento de la biblioteca, ordenados según estén en el lado *Cliente, Servidor de Autorización o Servidor de recursos.*

6.1.1 CLIENTE

clientConfig.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <OAuthClient>
3.       <AuthServerConfig>
4.            <AssertionType>urn:mace:rediris.es:papi</AssertionType>
5.            <GrantType>assertion</GrantType>
6.            <AuthServerURL> /url del AS token endpoint/ </AuthServerURL>
7.       </AuthServerConfig>
8.       <ResServerConfig>
9.            <RequestType>Form-Encoded_Body_Parameter</RequestType>
10.           <ResServerURL> /url del RS server endpoint/ </ResServerURL>
11.           <ResponseFormats>
12.                <Scope id="scope_de_prueba">
13.                     <FormatClass></FormatClass>
14.                     <FormatFile></FormatFile>
15.                </Scope>
16.           </ResponseFormats>
17.      </ResServerConfig>
18.      <ClientConfig>
19.           <ClientID>prueba_oauth</ClientID>
20.           <ClientSecret>clave_de_prueba_oauth</ClientSecret>
21.           <ErrorResponseType>HTML</ErrorResponseType>
22.           <DefaultScope>scope_de_prueba</DefaultScope>
23.           <DebugActive>TRUE</DebugActive>
24.           <StorageType>session</StorageType>
25.      </ClientConfig>
26. </OAuthClient>
```

Contiene la configuración del cliente, dividida en 3 partes:

- `AuthServerConfig` : configura el servidor de autorización en el cual el cliente ha sido registrado. Parámetros a configurar:

  - `AssertionType` - puede tomar dos valores:

    · ***urn:mace:rediris.es:papi*** , si la aserción que va a usar el cliente para obtener el token de acceso es de tipo PAPI.

· ***urn:oasis:names:tc:SAML:2.0:assertion*** , si la aserción que va a usar el cliente para obtener el token de acceso es de tipo SAML2.

- `GrantType` - debe de tener el valor ***assertion***.

- `AuthServerURL` - la URL del token endpoint del Servidor de Autorización.

- `ResServerConfig` : configura el Servidor de Recursos(RS) al cual el cliente, haciendo uso del token de acceso obtenido de un AS, va a pedir un recurso, determinado por su ***scope***.

- `RequestType` - puede tomar tres valores, que definen tres métodos de enviar el token de acceso (de tipo Bearer token) en la petición del recurso al RS:

· ***HTTP_Authorization_Header*** : hace la petición del recurso metiendo el token de acceso en el campo *Authorization* de las cabeceras http.

· ***Form-Encoded_Body_Parameter*** : hace la petición del recurso metiendo el token de acceso en el Body (POST) de la petición http.

· ***URI_Query_Parameter*** : hace la petición del recurso metiendo el token de acceso en la URI de la petición HTTP (GET).

- `ResServerURL` - la URL del server endpoint del Servidor de Recursos.

- `ResponseFormats` - Define las clases que, por reflexión, darán formato al recurso devuelto por el RS. Cada **scope**, definido por su identificador **id**, debe tener un `FormatClass`, con el nombre de la clase que da formato a ese **scope**, y un `FormatFile` con el nombre del archivo que contiene la clase.

- `ClientConfig` : configuración del cliente.

- `ClientID` - Identificador del cliente. Su valor debe ser suministrado por el AS en el momento del registro del cliente en él.

- `ClientSecret` - Clave del cliente.

- `ErrorResponseType` - Puede ser **HTML** o **JSON.**

- `DefaultScopeStorageType` - El scope por defecto que usa el cliente para pedir un token/recurso.

- `StorageType` - La forma en que guarda el cliente el token. Todas ellas usan el uid del cliente como clave de almacenamiento. Puede ser:

  · **sesión**, si se quiere guardar el token en la variable $_SESSION.

  · **db**, si se quiere guardar el token en una base de datos tipo dba.

  · **file**, si se quiere guardar el token de acceso en un fichero de texto plano.

- DebugActive - Si está a **TRUE**, se guarda una traza de la petición.

6.1.2 SERVIDOR DE AUTORIZACIÓN (AS)

clientKeys.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Clients>
3.       <Client id="prueba_oauth">
4.            <TokenLifetime>3600</TokenLifetime>
5.            <Key>clave_de_prueba_oauth</Key>
6.            <GenerateAccessTokenMethod>AS</GenerateAccessTokenMethod>
7.            <AllowedScopes>
8.                 <Scope id="scope_de_prueba"/>
9.                 <Scope id="scope_de_prueba2"/>
10.           </AllowedScopes>
11.      </Client>
12.      <Client id="prueba_oauth_SAML">
13.           <TokenLifetime>3600</TokenLifetime>
14.           <Key>clave_de_prueba_oauth_SAML</Key>
15.           <GenerateAccessTokenMethod>AS</GenerateAccessTokenMethod>
16.           <AllowedScopes>
17.                <Scope id="scope_de_prueba_SAML"/>
18.           </AllowedScopes>
19.      </Client>
20. </Clients>
```

Contiene la configuración de todos los clientes registrados en el Servidor de Autorización:

- id del Client - Es el ClientID guardado en la configuración de ese cliente.

- TokenLifetime -Tiempo en segundos para el cual el token es válido.

- `Key` - El `ClientSecret` guardado en la configuración del cliente.

- `GenerateAccessTokenMethod` –

  · ***AS*** : si el token de acceso es generado por el AS.

  · ***STS*** : si el AS delega en un STS la generación del token.

- `AllowedScopes` – Los scopes ,para los cuales, ese cliente tiene registrados que se pueda emitir un token.

serverKeys.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <AuthServer id="authserver_example" url="url AS token endpoint">
3.       <ResourceServers>
4.            <ResourceServer id="prueba_oauth">
5.                 <Scopes>
6.                      <Scope>scope_de_prueba</Scope>
7.                      <Scope>scope_de_prueba_SAML</Scope>
8.                 </Scopes>
9.                 <Key>oauth_server_key</Key>
10.           </ResourceServer>
11.           <ResourceServer id="prueba_oauth2">
12.                <Scopes>
13.                     <Scope>scope_de_prueba2</Scope>
14.                </Scopes>
15.                <Key>oauth_server_key2</Key>
16.           </ResourceServer>
17.           <ResourceServer id="prueba_oauth3">
18.                <Scopes>
19.                     <Scope>scope_de_prueba3</Scope>
20.                </Scopes>
21.                <Key>oauth_server_key3</Key>
22.           </ResourceServer>
23.      </ResourceServers>
24. </AuthServer>
```

Contiene la configuración de todos los Servidores de Recursos registrados en el Servidor de Autorización:

- `AuthServer` - contiene los atributos `id`, con el identificador del servidor de autorización, y `url`, con la URL del Servidor de Autorización.

- `ResourceServer` - contiene los `Scopes` soportados por cada Servidor de Recursos, junto con un `Key`, que servirá como clave para generar el valor cifrado del `id` del Servidor de Autorización

policies.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <AssertionList>
3.       <Assertion type="papi">
4.            <Policies scope="scope_de_prueba">
5.                 <TokenFormat>
6.                      <format>%sHO%</format>
7.                      <format>%mail%</format>
8.                      <format>%uid%</format>
9.                 </TokenFormat>
10.                <Policy>
11.                     <Attributes check="all">
12.                          <Attribute name="mail" value=""/>
13.                          <Attribute name="uid" value=""/>
14.                     </Attributes>
15.                     <Attributes check="any">
16.                          <Attribute name="sHO" value=""/>
17.                          <Attribute name="sHO" value=""/>
18.                     </Attributes>
19.                     <Attributes check="none">
20.                          <Attribute name="mail" value=""/>
21.                          <Attribute name="uid" value=""/>
22.                     </Attributes>
23.                </Policy>
24.           </Policies>
25.           <Policies scope="scope_de_prueba2">
26.                <TokenFormat>
27.                     <format>%sHO%</format>
28.                </TokenFormat>
29.                <Policy>
30.                     <Attributes check="any">
31.                          <Attribute name="sHO" value=""/>
32.                     </Attributes>
33.                     <Attributes check="all">
34.                          <Attribute name="mail" value=""/>
35.                     </Attributes>
36.                </Policy>
37.           </Policies>
38.      </Assertion>
```

```
39.      <Assertion type="saml2">
40.           <Policies scope="scope_de_prueba_SAML">
41.                <TokenFormat>
42.                     <format></format>
43.                     <format></format>
44.                </TokenFormat>
45.                <Policy>
46.                     <Attributes check="any" >
47.                          <Attribute name=" " value=""/>
48.                     </Attributes>
49.                     <Attributes check="all" >
50.                          <Attribute name="" value=""/>
51.                     </Attributes>
52.                     <Attributes check="none" >
53.                          <Attribute name="" value=""/>
54.                     </Attributes>
55.                </Policy>
56.           </Policies>
57.      </Assertion>
58. </AssertionList>
```

Contiene las políticas, según el scope, que se deben cumplir para que el Servidor de Autorización pueda emitir un token válido. Esta dividida por tipo de aserción(papi o saml2), debiendo de crearse dos políticas distintas para un mismo scope, una por cada tipo de aserción:

- `TokenFormat` - son atributos, extraídos de la aserción, necesarios para obtener el recurso asociado al scope. Estos atributos serán parte del token.

- `Policy` - es la política a seguir para el cada scope. Los atributos `check` pueden ser:

  · **any** : al menos debe de venir en la aserción un atributo `name` cuyo valor sea `value`.

· **all** : todos los atributos que vengan en la aserción con nombre `name` deben de tener el valor `value`.

· **none** : no puede venir en la aserción ningún atributo con nombre `name` cuyo valor sea `value`.

6.1.3 SERVIDOR DE RECURSOS(RS)

asKeys.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!--Registered Auth Servers-->
3. <AuthServers>
4. <AuthServer id="authserver_example" url="url token endpoint AS">
5. <Key>oauth_server_key</Key>
6. </AuthServer>
7. </AuthServers>
```

Contiene la información sobre los Servidores de Autorización registrados, para los cuales el Servidor de Recursos puede decodificar los tokens de acceso que le llegan de un cliente, pudiendo verificar si son válidos o no:

- `AuthServer` – contiene el identificador `id` del Servidor de Autorización y la `url` de su token endpoint. Además tiene una `key` que

servirá para verificar que el token de acceso ha sido emitido por un Servidor de Autorización registrado.

resourceClasses.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Resources>
3.       <Scope id="scope_de_prueba">
4.            <TokenFormat>
5.                 <format>%sHO%</format>
6.                 <format>%mail%</format>
7.                 <format>%uid%</format>
8.            </TokenFormat>
9.            <ResourceClass>Resource</ResourceClass>
10.           <ResourceFile>Resource.php</ResourceFile>
11.      </Scope>

12.      <Scope id="scope_de_prueba_SAML">
13.           <TokenFormat>
14.                <format></format>
15.                <format>format>
16.           </TokenFormat>
17.           <ResourceClass>Resource</ResourceClass>
18.           <ResourceFile>Resource.php</ResourceFile>
19.      </Scope>
20. </Resources>
```

Contiene los atributos, que deben venir en el token, para poder acceder al recurso definido por cada scope, además de la clase que se usará para dar formato a ese recurso.

6.2 USO DEL LA BIBLIOTECA OAUTH2LIB

```
1.  <?php
2.  include 'oauth_client/src/OAuth.php';
3.  include 'phpPoA-2.4/PoA.php';
4.  $poa = new PoA('','');
5.  $oauth = $poa->authenticate();
6.  $attrs = $poa->getAttributes();
7.  $filePAPI =
    dirname(dirname(__FILE__)).'oauth2lib/oauthclient/config/clientCon
    fig.xml';
8.  $oauthPAPI = new OAuth($filePAPI);
9.  $oauthPAPI->requestAccessToken($attrs);
10. $oauthPAPI->requestResource($oauthPAPI->getRS(), $oauthPAPI-
    >getRequest_type());
11. $contentPAPI = $oauthPAPI->getResource();
12. ?>
```

Ejemplo de uso de oauth2lib con aserciones PAPI

1. Una vez obtenida la aserción(mediante phpPoA para el caso PAPI), se llama al constructor de la clase OAuth, el cual, entre otras cosas, carga el archivo de configuración que se le pasa como parámetro.

2. Se llama al método de clase ***requestAccessToken***, el cual recibe como parámetro un array con los atributos de la aserción PAPI(obtenido a través del método ***getAttributes*** de la clase PoA de phpPoA) o una cadena con la aserción SAML2.

3. Este método, internamente, se encargará de hacer la petición del token de acceso al Servidor de Autorización que tenga configurado ese cliente. Aunque primero comprobará que el usuario que va a hacer la petición no tenga ya un token de acceso guardado(en la variable $_SESSION o en una base de datos dba), en cuyo caso se omite la petición al AS. En caso de haber petición, seguirá los siguientes pasos:

    a. 3.1. Si el usuario no tiene token de acceso guardado, se usa el método de la clase OauthClient, ***doAccessTokenRequest***, que recibe como parámetros el AS, el scope, la aserción, el assertion_type y el grant_type, todos ellos sacados de la configuración del cliente.

    b. *3.2.* Este método primero comprueba que la petición se hace de manera segura, mediante protocolo https, genera la petición y la manda mediante cURL al AS. Una vez obtenida la respuesta, se procesa y se guarda el token en el atributo de clase *access_token.*

4. Una vez obtenido el token de acceso, se pasará a pedir el recurso mediante la función **requestResourceRS**, el cual recibe como parámetros la URL del Servidor de Recursos y el tipo de petición que se hará(puede ser *HTTP_Authorization_Header, URI_Query_Parameter o Form---Encoded_Body_Parameter).* El Servidor de Recursos comprobará que el token de acceso es válido y devolverá el recurso solicitado, al cual se puede acceder a través del método getResource de la clase OAuth.

El uso con aserciones SAML2 es casi idéntico, lo único que puede variar es el método de obtención de la aserción SAML2, para lo cual no es necesario el uso de la biblioteca phpPoA.

7. CONCLUSIONES

Los pasos que se han seguido a la hora de elaborar el proyecto sirven como resumen del mismo:

1. Primero se realizó un estudio del protocolo OAuth, en qué contexto se aplicaba, qué tecnologías requería y qué características especificaba.
2. Una vez hecho eso, se pasó a estudiar de qué modo el protocolo era aplicable a las características del Servicio de Identidad de RedIRIS.
3. Una vez situado en contexto, el siguiente paso era observar como estaba construida la biblioteca ya implementada dentro de RedIRIS, y establecer unos objetivos para su mejora y adaptación a las últimas versiones del protocolo, así como detectar los fallos que pudiera tener la misma.
4. Realizar las mejoras en la biblioteca establecidas en los objetivos y corregir los fallos encontrados.

Debido a que la implementación de los servicios de identidad digital de RedIRIS

son, en su mayoría, desarrollados en código PHP, se decidió implementar en este lenguaje la solución.

Otro aspecto interesante del proyecto es la posibilidad que tecnologías innovadoras en identidad digital como OAuth ofrecen a un público menos especialista. Gracias a la simplicidad a la hora de implantar estas soluciones en el lado del cliente es posible que aplicaciones de bajo presupuesto puedan incorporar sistemas de acceso a recursos protegidos que con otro tipo de tecnología implicarían un esfuerzo mayor a los desarrolladores de las mismas.

Este motivo es uno de los cuales han propiciado el auge de una nueva generación de herramientas en identidad digital, la cual pretende implementar sistemas de autenticación y autorización basados en tecnologías que garanticen la seguridad de las comunicaciones, pero que no resulten excesivamente complejas ni ocupen muchos recursos.

Debido a la forma modular en que está diseñada la biblioteca, sería muy fácil en el futuro añadir nuevas funcionalidades, tales como el soporte para otro tipo de aserciones, la inclusión en el flujo de nuevos tipos de *authorization grants*.

Uno de los aspectos observados al realizar este proyecto ha sido la necesidad de documentarse e investigar antes de realizar ninguna implementación. Si bien es importante llevar a la práctica los conceptos asimilados, más importante es realizar un estudio exhaustivo de las tecnologías que se van a utilizar, para así poder tomar las decisiones correctas cuando sea necesario.

Además de esto se ha visto no sólo la necesidad de documentarse, si no la actualización permanente durante el transcurso de un proyecto, especialmente cuando las tecnologías que están utilizándose son tan novedosas.

BIBLIOGRAFÍA

- *Borrador de IETF* – **The OAuth 2.0 Authorization Protocol draft-ietf-oauth-v2-22** http://tools.ietf.org/html/draft-ietf-oauth-v2-22

- *Borrador de IETF* - **OAuth 2.0 Assertion Profile draft-ietf-oauth-assertions-01** http://tools.ietf.org/html/draft-ietf-oauth-assertions-01

- *Borrador de IETF* - **SAML 2.0 Bearer Assertion Profiles for OAuth 2.0 draft-ietf-oauth-saml2-bearer 10** http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-10

- *Borrador de IETF* - **The OAuth 2.0 Authorization Protocol: Bearer Tokens draft-ietf-oauth-v2-bearer-14** http://tools.ietf.org/html/draft-ietf-oauth-v2-bearer-14

- Borrador IETF - **JSON Web Token (JWT) draft-ietf-oauth-json-web-token-03** http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

- **Servicio de Identidad de RedIRIS (SIR)** http://www.rediris.es/sir/

- **Página oficial de Oauth** http://oauth.net

- **Página oficial de PAPI** http://papi.rediris.es

- **Página oficial de SAML** http://saml.xml.org

- **Integración del protocolo OAuth con el Servicio de Identidad de RedIRIS** http://www.rediris.es/ptyoc/res/dl22.html.es

GLOSARIO DE TÉRMINOS

Es necesaria la definición de una serie de conceptos antes de entrar en el estudio del protocolo:

- **AUTENTICACIÓN:** Procedimiento por el cual un usuario se identifica en una organización. En caso positivo obtiene unas credenciales con la información asociada a su identidad.
- **AUTHORIZATION GRANT:** Término abstracto usado para describir las credenciales intermedias que representan la autorización del propietario del recurso.
- **AUTORIZACIÓN:** Procedimiento por el cual, una vez autenticado el usuario y dependiendo de las credenciales o atributos que tenga, se le podrá permitir el acceso a recursos.
- **CLIENTE:** Aplicación web que utiliza OAuth para acceder a un recurso en nombre de su propietario. Formalmente, se trata de un cliente HTTP capaz de hacer peticiones OAuth autenticadas.
- **CREDENCIALES:** Incluirán información acerca de si se ha realizado correctamente la autenticación y los atributos asociados al usuario. Intenta reducir información sensible (DNI, nombre, apellidos, etc.) y reflejar campos más generales como tipo de usuario, nombre de usuario, etc. Podrán ir cifradas de modo que sólo los proveedores de identidad y los proveedores de servicios puedan leerlas.

- **IDENTIDAD FEDERADA:** la identidad federada es una de las soluciones para abordar la gestión de identidad en los sistemas de información basados en entornos federados.
- **IETF:** es el Internet Engineering Task Force(en español, grupo especial sobre ingeniería en internet) es una organización internacional abierta de *normalización*, que tiene como objetivo contribuir a la ingeniería de internet, actuando en diversas áreas. Uno de los proyectos de normalización que actualmente está en desarrollo es el protocolo OAuth 2.0, el cual se va a tratar en este proyecto.
- **INSTITUCIÓN:** Cada uno de los participantes en de entornos federados, típicamente con relaciones académicas, institucionales o científicas.
- **JWT:** JSON Web Token, es un medio de representación de las reclamaciones(*claims*) que se transfieren entre dos partes. Estas *claims* se codifican en un objeto de tipo JavaScript Notation, JSON, y se firman digitalmente.
- **OAUTH :** Protocolo de autorización que habilita a aplicaciones de terceros a obtener acceso limitado a un servicio HTTP, ya sea con una autorización expresa del propietario de un recurso o la propia aplicación en nombre de este.
- **PAPI:** Software, desarrollado inicialmente por RedIRIS, que permite desplegar una federación de identidad digital mediante un sistema de delegación en la autorización. Es ampliamente utilizado por las instituciones académicas y científicas españolas.

- **phpPoA:** Biblioteca, en PHP, que permite conectar cualquier aplicación con el Servicio de Identidad de RedIRIS (SIR), previo registro de la aplicación.
- **PROPIETARIO DEL RECURSO:** Entidad capaz de acceder y controlar recursos protegidos.
- **PROVEEDOR DE IDENTIDAD(IdP):** Entidad que realiza la autenticación del usuario y emite sus credenciales.
- **PROVEEDOR DE SERVICIOS(SP):** Entidad que, una vez comprobadas las credenciales de un usuario y, dependiendo de estas, dará acceso a un recurso o lo denegará.
- **RECURSO:** Elemento al cual se quiere acceder. Podrá estar protegido.
- **SAML:** Security Assertion Markup Language (Lenguaje de Marcas de Aserciones de Seguridad). Estándar basado en XML para el intercambio de datos correspondientes a autenticación y autorización entre dominios de forma segura.
- **SCOPE:** es el ámbito para el cual un token de acceso es válido, el alcance de hasta que recursos puedo obtener con un token determinado.
- **SERVICIO WEB(WS):** Pieza de software que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
- **SERVIDOR DE AUTORIZACIÓN(AS):** El servidor que emite tokens de acceso a los clientes que lo soliciten, los cuales, previamente, deben haber autenticado al propietario del recurso y haber obtenido su autorización.

- **SERVIDOR DE RECURSOS(RS):** El servidor que aloja los recursos protegidos, capaz de aceptar y responder a peticiones de recursos protegidos que contengan tokens de acceso.
- **STS:** Software basado en servicios web o sitios web responsable de emitir tokens de seguridad. En su uso más común, un cliente solicita acceso a una aplicación segura. La aplicación no valida la identidad del cliente por si misma, sino que redirige al cliente a un STS pasándole las credenciales. El STS verifica las credenciales enviadas por el cliente, y en caso de ser correctas, emite un token de seguridad, el cual proporciona la prueba de que el cliente se ha autenticado satisfactoriamente en el STS. Por último, el cliente es redirigido a la aplicación a la que, en un principio, se intentó acceder.
- **TOKEN DE ACCESO:** Cadena de caracteres (String), opaca al cliente, que se usa como credenciales para acceder a recursos protegidos y que representan una autorización que es emitida, por un Servidor de Autorización, a un Cliente. Cada token de acceso tiene un Scope y un tiempo de vida, determinados por el propietario del recurso e impuestos tanto al Servidor de Autorización como al Servidor de Recursos.
- **USUARIO:** Entidad, típicamente una persona física, que actúa como propietario del recurso e interactúa con el cliente para la obtención de tokens de acceso para acceder a recursos protegidos.