UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERíA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Estudio y puesta en marcha de una infraestructura de gestión de identidad federada basada en SAML 2.0

Autora: Rosa Mª Sánchez Guerrero
Tutora: Florina Almenárez Mendoza

4 de diciembre de 2009

Agradecimientos

A mis padres y a mi hermana, por creer y confiar siempre en mí; y porque sin vosotros nunca habría conseguido llegar a este momento. Gracias a mis padres por la educación que me habéis dado y por vuestro apoyo en los buenos y en los malos momentos. Gracias a mi hermana, por estar ahí siempre que te he necesitado.

A mis amigas desde siempre, Alejandra y Bea. Gracias por vuestra amistad, por los buenos momentos que hemos compartido durante todos estos años y por animarme y apoyarme cuando lo he necesitado.

A mi amiga Natalia. Gracias por todo lo que me has aportado a lo largo de estos años, tanto a nivel académico, por haber sido para mí una excelente compañera; como a nivel personal, porque en ti he encontrado una gran amiga y persona. Espero que siempre estés muy presente en mi vida.

A mis compañeros de la Universidad, Belén, Carlos, Dani, Fer, Javi, Javo, Kiko, Laura, Luis, Manu, Miki, Raúl, Patricia, Rober y Santi, porque en vosotros he encontrado además unos grandes amigos y porque cada uno, habéis contribuido de alguna manera a que llegue hasta aquí. Gracias por los inolvidables momentos que hemos pasado juntos y por el recuerdo que me llevo de mi estancia en la Universidad.

En especial, a Rober. Por tu apoyo incondicional, tu amor y tu cariño. Gracias por tu infinita paciencia, por darme ánimos en aquellos momentos en los que he querido rendirme; y por darme ilusión y fuerzas para conseguir lo que me propongo.

A Nieves, por tratarme como una más de la familia.

A mi tutora Florina, primero por la confianza que depositaste en mí para la realización de este proyecto. Por por tu inestimable ayuda, consejos y supervisión durante todos estos meses. Gracias por estar siempre disponible para cualquier cosa y por orientarme en la búsqueda de un trabajo bien hecho.

A mis compañeros de grupo de investigación: Patri, Davide, Fabio y los profesores Andrés Marín y Daniel Díaz. Primero, gracias por haber confiado en mí. Gracias por vuestras aportaciones e ideas, y por la valiosa ayuda que me habéis prestado durante todos estos meses.

A mis compañeros del "A01", por los buenos ratos que he pasado con vosotros a lo largo de estos meses.

Resumen

Actualmente el uso de infraestructuras que permitan realizar tareas de gestión de identidad se ha convertido en un requisito indispensable para la mayoría de las empresas. Un sistema de gestión de identidad permite cubrir las necesidades clave de seguridad de una organización, otorgándole a la vez la libertad de crecer tan rápido como le permita su negocio. Los sistemas de gestión de identidad separan las tareas de provisión de servicios, de aquellas que tienen como propósito gestionar identidades. De este modo, permite liberar a los proveedores de servicios de la gestión de datos relativos al acceso del usuario. Por lo que no tienen que preocuparse de llevar a cabo tareas tan tediosas como el almacenamiento de contraseñas y certificados de los usuarios que acceden al sistema. Además, desde el punto de vista del usuario, este tipo de sistemas tiene como objetivo la facilidad de uso, por lo que permiten realizar procesos de inicio y cierre de sesión únicos. Para ello, éste sólo necesita autenticarse en un proveedor de identidad, que comunicará al resto de los proveedores con los que tenga establecida una relación de confianza, que el usuario ha sido autenticado.

En los últimos años han surgido diversas iniciativas que tienen como objetivo definir marcos de trabajo basados en gestión de identidad. Entre ellas, podemos destacar el lenguaje de asertos y la infraestructura definida por OASIS, SAML (*Security Assertion Markup Language*), las iniciativas desarolladas por Liberty Alliance, WS-Federation y OpenID. Otra iniciativa a destacar es la de Shibboleth, que está basada en SAML 2.0. De ellas, la única que se encuentra estandarizada hasta el momento es SAML. Todos estos marcos de trabajo tienen como objetivo reducir la complejidad de los proveedores de servicios y permitirles centrarse en su núcleo, a la vez que proporcionan una experiencia satisfactoria a los usuarios que acceden a sus servicios.

El objetivo final de este proyecto, es realizar un estudio práctico acerca de la especificación SAML 2.0 y de las funcionalidades que proporcionan los distintos perfiles y casos de uso para gestión de identidad federada contemplados en el estándar. Dicho estudio incluye la puesta en marcha de una infraestructura de gestión de identidad basada en desarrollos de código abierto como son ZXID y Lasso. Además se han desarrollado nuevos componentes para explorar funcionalidades no cubiertas en las implementaciones utilizadas. En concreto, nos hemos centrado en los perfiles de *Single-Sign-On* y *Single-Logout*, con el fin de hacer pruebas de integración e interoperabilidad de los distintos proveedores que constituyen la plataforma de gestión de identidad desplegada en este proyecto.

Abstract

Using identity management infrastructures is usual nowadays because it has become a prerequisite for most companies. An identity management system can meet the key needs of an organization's security, while giving the freedom to grow as fast as his business allows. This systems separate identity management tasks from of service provisioning. Thus, service providers are freed from managing the data for user access. Therefore, they don´t have to worry about carrying out tedious tasks such as storing passwords and certificates of the users accessing the system. Moreover, from the user´s perspective, this type of systems is aimed at ease of use, so it allows single sign on and single logout. Thus, the user only needs to authenticate to an identity provider, who will communicate to other providers with which it has established a relationship of trust that the user has been authenticated.

There are various initiatives currently that aim at defining frameworks based on identity management. Among them, we can highlight the language of assertions and infrastructure defined by OASIS, SAML (Security Assertion Markup Language), the initiatives Liberty Alliance, WS-Federation and OpenID. Another initiative worth noting is that of Shibboleth, which is based on SAML 2.0. Of these, the one that is standardized so far is SAML. All these frameworks aim to reduce the complexity of service providers and enabling them concentrate on their core business, while providing a satisfactory experience to users who access their services.

The main aim of this project is to carry out a study on SAML version 2.0 and functionalities provided by different profiles and use cases for federated identity management covered by the standard. Such study includes the deployment of the infrastructure based on open source developments such as ZXID and Lasso. In addition, new components have been developed in order to cover functionalities that are not involved in the used implementations. Specifically, we have focused on the profiles of Single Sign On (SSO) and Single Logout (SLO) to perform tests of integration and interoperability between providers that are part of the identity management platform deployed in this project.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación del proyecto

En los últimos años, el crecimiento de las empresas y el incremento en las relaciones de negocio, que éstas han experimentado en diversos sectores como son: la administración electrónica, servicios financieros, proveedores de servicios *online*, telecomunicaciones y las industrias de turismo y transporte; han provocado la necesidad de mejorar y facilitar las relaciones de confianza y seguridad, entre las distintas empresas que colaboren, para el intercambio de información e identidades. Es en este momento, cuando la gestión de identidad federada se convierte en un concepto clave. Los modelos de gestión de identidad federada permiten separar las tareas de gestión de identidad de las de provisión de servicios, de modo que reducen la complejidad de los proveedores de servicios y permiten a las empresas centrarse en su núcleo de negocio.

La información de identidad se encuentra en el núcleo de la relación de una compañía con sus clientes, socios y otras empresas, y es imprescindible para sostener sus modelos de negocio. Desde un punto de vista técnico, se relaciona con la seguridad de red, la provisión del servicio, la gestión de clientes, la vinculación entre recursos de identidad distribuidos (federación de identidades) y la provisión de Servicios Web.

Por otro lado, desde la perspectiva del usuario, este incremento de servicios, especialmente notable en Internet, hace que acciones tan habituales como reservar un vuelo o hacer una compra, impliquen que el usuario tenga un número considerable de cuentas abiertas, con múltiples identidades y diferentes mecanismos de autenticación. Los diferentes servicios requerirán distintos niveles de seguridad durante la autenticación, por ejemplo, no se exige el mismo nivel de seguridad

para acceder a un blog que para realizar una transacción bancaria.

Para proporcionar una experiencia más satisfactoria a los usuarios que acceden a estos servicios, las infraestructuras basadas en gestión de identidad disponen de mecanismos que hacen posible el establecimiento y finalización de sesiones de manera sencilla y transparente al usuario, a través de los populares casos de uso de *Single Sign-On* (SSO) y *Single Logout* (SLO). El proceso de *Single Sign-On* permite al usuario autenticarse frente a un único proveedor de identidad, que es el encargado de mediar con los sitios federados para comunicar la identidad del usuario sin necesidad de requerir nuevas autenticaciones del usuario.

De este modo, la gestión de identidad tiene como objetivo facilitar en gran medida la colaboración con terceros, proporcionar privacidad y mejorar la utilización y la seguridad de los servicios.

Como respuesta a la problemática de federación de identidad, han surgido diversas iniciativas empresariales y de investigación, como SAML, Liberty Alliance, OpenID, WS-Federation, etc, para establecer un marco de trabajo que ofrezca, a los desarrolladores de servicios, mecanismos básicos de seguridad e identidad, con el objetivo de que el proveedor de servicios se centre en su núcleo de negocio y no deba preocuparse por llevar políticas de gestión de los datos de acceso de usuarios, de autenticación o de autorización.

Sin embargo, de estas iniciativas, la única que se encuentra estandarizada por OASIS [1] hasta el momento, es SAML 2.0. Además, se trata del protocolo más completo y más extensamente implementado dentro de este ámbito.

Hoy en día, importantes empresas de todo el mundo, entre las que podemos citar Google, IBM, *Sun Microsystem*, etc, ofrecen soluciones que están basadas en SAML 2.0 o son compatibles con esta especificación. Sin embargo, la mayor dificultad que presentan este tipo de sistemas, radica en que se requiere confianza en los *partners* y a día de hoy no existen productos fáciles de implantar, debido a que las relaciones de confianza entre los distintos proveedores que componen un escenario de gestión de identidad basado en SAML, típicamente se realiza de manera estática.

La motivación de este proyecto viene dada por el estudio de la especificación SAMLv2.0, pues se trata de un lenguaje extensible y es una especificación abierta. El hecho de que cada cierto tiempo se publiquen nuevas revisiones de esta iniciativa, permite mostrar que el tema de la **Gestión de Identidad** se encuentra en plena evolución. Por lo que es necesario continuar realizando estudios acerca de las novedades incorporadas en este estándar y en qué medida esto puede afectar a la compatibilidad entre distintas herramientas que inicialmente parten con el mismo objetivo.

Para investigar acerca de la funcionalidad de los distintos perfiles y casos de uso que se contemplan en este estándar, hemos desplegado la plataforma de gestión de identidad propuesta a lo largo de este proyecto.

1.2. Objetivos

El objetivo general de este proyecto es poner en marcha una plataforma de gestión de identidad federada basada en las especificaciones de SAML que garantice los aspectos de seguridad básicos. A continuación se enumeran los objetivos específicos que debe cumplir este proyecto:

- Llevar a cabo un estudio de las principales herramientas de código abierto disponibles que implementen parte de la especificación SAMLv2. Estas herramientas nos permitirán:
  - Poner en marcha los elementos básicos de un sistema basado en gestión de identidad: un proveedor de servicios y un proveedor de identidad.
  - Desarrollar nuestro propio proveedor de servicios, a fin de explorar el estándar en mayor profundidad y probar nuevas funcionalidades.
- Probar el funcionamiento de distintas variantes de los mecanismos de inicio y cierre de sesión únicos, a través del sistema desplegado.
- Evaluar la interoperabilidad de los distintos proveedores que constituyen el sistema. Es necesario que dichos proveedores sean compatibles, a fin de que el usuario pueda llevar a cabo un inicio o un cierre de sesión único, desde cualquiera de ellos, de forma transparente.
- Ofrecer una gestión de usuarios eficiente.
- Desplegar un sistema de gestión de identidad que resulte fácil de utilizar para los usuarios.
- Comprobar que los proveedores sean accesibles desde dispositivos móviles y máquinas remotas a través de distintos tipos de navegadores web existentes.

1.3. Contenido de la memoria

El resto de la memoria está estructurada como sigue:

- El Capítulo 2 introduce las diferentes tecnologías que constituyen el contexto del proyecto. Se introducen los principales aspectos de los servicios básicos de seguridad requeridos en la mayoría de las aplicaciones que funcionan en red. También se detallan las principales características de las infraestructuras de clave pública y firma XML, en las que se basan algunas especificaciones de gestión de identidad. A continuación, se realiza una revisión acerca de las principales iniciativas de federación de identidad. Finalmente, se incluye una descripción del protocolo LDAP y de los lenguajes de programación utilizados para el desarrollo del proyecto.

- En el Capítulo 3 se realiza una descripción general del sistema, de los requisitos funcionales y se lleva a cabo un estudio de las herramientas necesarias para proporcionar estos requisitos. Además, se presentan detalles de la arquitectura, proporcionando una explicación de los módulos involucrados, funcionalidades y relaciones.

- En el Capítulo 4 se describe el proceso de puesta en marcha de la infraestructura de gestión de identidad, y de los distintos perfiles definidos en SAML que podremos probar.

- En el Capítulo 5 se explican los detalles de implementación de un proveedor de servicios y se incluyen diversos diagramas para clarificar el funcionamiento.

- En el Capítulo 6 se documentan las pruebas realizadas para los proveedores desplegados en el capítulo 4 y el proveedor de servicios desarrollado en el capítulo 5. Se reflejan las pruebas de los distintos módulos del sistema, incluyendo el resultado y observaciones de interés. Además de pruebas del funcionamiento, también se han realizado *tests* de interoperabilidad y pruebas con distintos navegadores.

- En el Capítulo 7 describimos las distintas fases de desarrollo del proyecto, explicando las tareas realizadas en cada una, los problemas encontrados y los resultados obtenidos.

- En el Capítulo 8 se resumen las principales conclusiones del proyecto realizado y se enumeran una serie de posibles líneas de trabajo futuro.

Como parte de la memoria hemos incluido una serie de apéndices para aclarar y completar algunos aspectos relacionados con la solución propuesta en este proyecto:

- En el apéndice A se detalla el presupuesto del proyecto.

- En el apéndice B encontramos una descripción acerca de los pasos necesarios para llevar a cabo la instalación de las distintas librerías utilizadas para el desarrollo de este proyecto. En este apéndice, también incluyen los pasos necesarios para instalar y configurar un servidor LDAP basado en OpenLDAP.
- En el apéndice C explicamos cómo crear una Autoridad de Certificación y generar certificados digitales utilizando OpenSSL.
- En el apéndice D incluimos un glosario de términos que aparecen a lo largo de esta memoria.

Capítulo 2

Estado del arte

2.1. Introducción

La integración de mecanismos de seguridad basados en gestión de identidad es uno de los factores claves para el desarrollo y la implantación de las tecnologías y modelos orientados a servicios. Aunque existen varios modelos y estándares para tratar el tema de la gestión de identidad, la integración efectiva de dichos estándares en entornos de desarrollo compuestos por múltiples dominios de confianza con distintos proveedores de identidad, es un tema abierto donde las soluciones están por llegar.

La federación de identidad tiene como objetivo principal compartir y distribuir información relativa de identidad y atributos a través de diferentes dominios de confianza, según ciertas políticas establecidas. El modelo de federación permite a los usuarios el acceso a recursos de otro dominio de una forma segura, sin necesidad de que los procesos redundantes den acceso al usuario. Dentro de esta problemática, existen diversas iniciativas que permiten federación de identidad, de las cuales cabe destacar, el lenguaje de asertos y la infraestructura definida por OASIS, SAML [2], Liberty Alliance [3] [5], y las especificación definidas por la iniciativa de Internet2 [6], Shibboleth [7]. Otras iniciativas son, OpenId [8] y WS-Federation [9].

Antes, es necesario conocer los servicios básicos de seguridad requeridos en la mayoría de las aplicaciones que funcionan en red. Además, se introducen las infraestructuras de clave pública y firma XML. Asimismo, teniendo en cuenta los objetivos y las características del presente proyecto, resulta imprescindible proporcionar al lector una perspectiva del protocolo LDAP, utilizado para mejorar las prestaciones de almacenamiento de usuarios ofrecidas por una de las entidades

de nuestro sistema (el proveedor de identidad), junto con una descripción de las características fundamentales de los lenguajes de programación utilizados para el desarrollo de este proyecto.

2.2. Tecnologías de Seguridad Base

2.2.1. Nociones básicas

Los principales servicios básicos de seguridad que requieren los dominios son:

- **Autenticación**. Es un mecanismo que permite garantizar que un usuario que envía un mensaje es realmente quien dice ser. Por medio de la autenticación pueden evitarse ataques de suplantación de identidad.
- **Autorización**. Este servicio es el encargado de controlar el acceso a los recursos (información, comunicación, entidades físicas, etc.), permitiendo que el usuario pueda acceder a ciertos servicios.
- **Integridad**. Este servicio se encarga de garantizar que la información no ha sido modificada.
- **Confidencialidad**. Es el servicio que permite que la información no esté disponible o sea accesible para entidades no autorizadas.
- **Disponibilidad**. Es necesario que los recursos del sistema estén disponibles a las entidades autorizadas cuando los necesiten.
- **No repudio**. Este servicio proporciona protección a un usuario frente a que otro usuario niegue posteriormente que en realidad se realizó cierta comunicación. Por ejemplo, garantizar que el emisor de un mensaje no podrá, posteriormente, negar haberlo enviado, mientras que el receptor no podrá negar haberlo recibido.
- **Auditoría**. Se define como el estudio que comprende el análisis y gestión de sistemas para identificar y corregir las diversas vulnerabilidades que pudieran presentarse en una revisión exhaustiva de las estaciones de trabajo, redes, servidor, etc.

Tanto estos servicios como la provisión de identidad pueden ser proporcionados utilizando criptografía simétrica y/o asimétrica. La criptografía simétrica se basa en un secreto compartido que debe ser distribuido de manera segura, por

ejemplo, contacto físico entre usuarios; mientras que la criptografía asimétrica se basa en una pareja de claves complementarias denominadas clave privada y clave pública. Esta última, también conocida como criptografía de clave pública fue introducida en 1976 por los matemáticos Whitfield Diffie y Martin Hellman [10]. La clave pública puede ser distribuida abiertamente, sin embargo, la clave privada únicamente puede ser conocida por el propietario de la misma. El funcionamiento se basa en que la información cifrada con la clave pública, sólo puede ser descifrada con la clave privada y viceversa.

Las aplicaciones principales de los algoritmos asimétricos son el cifrado de datos, para proporcionar confidencialidad, y las firmas digitales, que aseguran autenticación, no repudio e integridad del mensaje.

Si se desea enviar información confidencial a un usuario, se le enviará la información cifrada con su clave pública, de tal forma que sólo ese usuario, que posee la clave privada correspondiente, podrá descifrar la información. Por otra parte, si un usuario envía información cifrada con su clave privada, al descifrarla con su clave pública (acción que puede realizar cualquiera que conozca dicha clave), puede asegurarse que ha sido ese usuario quién envió la información, ya que sólo él posee la clave privada.

2.2.2. Infraestructura de clave pública X.509

La seguridad de una infraestructura de clave pública depende de la protección de la clave privada y la distribución de las claves públicas. En lo que se refiere a la protección de la clave privada, existen dispositivos especiales llamados *tokens* de seguridad que permiten el almacenamiento seguro de la misma. Un ejemplo de *token*, es una tarjeta inteligente. Para la distribución y gestión de las claves públicas se ha estandarizado la infraestructura de clave pública conocida como PKI (Public Key Infrastructure) [11], orientada a redes fijas.

La infraestructura de clave pública PKI es una combinación de software, tecnologías de cifrado, políticas, procedimientos y servicios que permiten proteger las transacciones de información en un sistema distribuido. Ofrece la seguridad básica requerida para llevar a cabo negocios electrónicos de tal manera, que los usuarios que no se conocen entre sí, o se encuentran muy alejados, pueden comunicarse con seguridad a través de una cadena de confianza.

X.509 es un estándar de la ITU-T [12] para PKI, en el que especifican aspectos tales como, los formatos para certificados de claves públicas y algoritmos de validación de caminos de certificación. A partir de este estándar, el grupo de trabajo del IETF, PKIX, ha desarrollado nuevos estándares y perfiles que direc-

cionan el uso de certificados X.509 en aplicaciones de Internet, tales como correo electrónico, comercio electrónico, etc.

La infraestructura PKI se basa en el uso de certificados digitales para asociar la identidad de un usuario con su clave pública. Además, se puede confiar en la identidad de los usuarios, ya que los certificados son emitidos por una entidad de confianza que avala sus datos. El propósito de PKI es permitir a los usuarios autenticarse frente a otros usuarios y usar la información de los certificados de identidad (por ejemplo, las claves públicas de otros usuarios) para cifrar y descifrar mensajes.

Algunas de las aplicaciones más frecuentes de PKI que cabe destacar, son autenticación de usuarios y sistemas, identificación del interlocutor (servidor/cliente seguro), cifrado de datos digitales, firmado digital de datos (documentos, correos, software, etc.), y aseguramiento de las comunicaciones.

2.2.2.1. Certificados X.509

Antes de que surgiera la idea de los certificados, era muy complicado garantizar que se habían establecido las relaciones de comunicación entre las entidades en un gran entorno de red o proporcionar la existencia de un depósito de confianza con todas las claves públicas utilizadas. Los certificados se inventaron con el objetivo de dar una solución al problema de distribución de claves públicas. Surge el concepto de Autoridad de Certificación (CA), que puede actuar como un tercero de confianza. En esta sección se exponen algunas nociones básicas que permitirán al lector entender en qué consisten los certificados X.509, así como el tipo de información que pueden contener los mismos.

Un certificado de clave pública es una declaración firmada digitalmente de una entidad, en la cual se especifica que la clave pública (y alguna otra información adicional) de otra entidad tiene un valor específico.

Los certificados X.509 contienen información relacionada con el usuario, la entidad emisora y el certificado en sí mismo, además de la firma digital del emisor. La sintaxis se define empleando el lenguaje ASN.1 (*Abstract Syntax Notation One*) y los formatos de codificación más comunes son DER (*Distinguished Encoding Rules*) o PEM (*Privacy-enhanced Electronic Mail*). DER define un formato de datos en binario, mientras que PEM está codificado en Base64.

Hasta el momento se han definido tres versiones del estándar X.509. En 1988 apareció X.509, versión 1, la cual está ampliamente desarrollada y es la más genérica. Posteriormente, con la segunda versión se introdujo el concepto de emisor e identificador únicos del sujeto, para manejar la posibilidad de reutilización

de los sujetos o nombres de emisor a través del tiempo. Finalmente, en 1996 aparece la versión 3, en la que cualquiera puede definir extensiones que pueden ser añadidas al certificado, tales como el uso permitido para la clave, ubicación de la lista de revocación (*CRL, Certificate Revocation List*), *KeyUsage* (limita el uso de las claves para propósitos específicos) y *AlternativeNames* (permite a otras identidades también estar asociados con esta clave pública, por ejemplo, direcciones de correo electrónico), etc.

A continuación se exponen los campos obligatorios de la última versión de un certificado X.509:

- **Versión** del estándar X.509 que aplica al certificado.
- **Número de serie** para distinguir el certificado de los demás, ya que es un identificador único para cada certificado.
- **Identificador del algoritmo** usado para generar la firma digital del certificado.
- **Nombre del emisor** del certificado de acuerdo con el estándar X.500 [14].
- **Período de validez** en el cual el certificado es válido.
- **Nombre del sujeto poseedor de la clave pública**, el cual es un nombre distinguible (DN, Distinguished Name) de acuerdo con el estándar X.500 o un nombre alternativo como una dirección de correo electrónico o una entrada de DNS [15][16].
- **Clave pública del sujeto** junto con el identificador del algoritmo usado para generar la pareja de claves.
- **Algoritmo** usado para firmar el certificado.
- **Hash** de la firma digital.
- **Firma** digital del certificado.

A continuación, se muestra un ejemplo de un certificado X.509 v3, emitido por Andrés Marín a Lasso IdP, el cual se trata de un Proveedor de Identidad del que se hablará más adelante en los capítulos 3 y 4. La clave pública es una clave RSA de 2048 (módulo y exponente público) y la firma ha sido creada con un resumen digital MD5 de la primera parte del certificado y cifrándola con la clave privada RSA de Andrés Marín. El tamaño del certificado es aproximadamente 2 KB. El certificado de Andrés Marín es un certificado raíz, es decir, un certificado auto-firmado, en el cual el emisor y el sujeto son iguales.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 12 (0xc)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ES, ST=Madrid, O=IT-UC3M, CN=Andres Marin
        Validity
            Not Before: Jun 17 08:42:39 2009 GMT
            Not After : Jun 17 08:42:39 2010 GMT
        Subject: C=ES, ST=Madrid, O=IT-UC3M, CN=Lasso IdP, CN=idp.gast.it.uc3m.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:c2:f4:8f:3a:f0:b7:ca:fe:f9:b8:d0:1b:b0:cf:
                    1d:a5:d0:65:6f:ef:b6:3e:ce:d0:96:4e:49:66:1f:
                    23:c2:ae:ee:ac:8d:74:81:d7:82:34:0a:e6:93:f6:
                    70:10:89:cc:1a:e0:93:a9:2b:f3:93:cb:55:55:1c:
                    a1:49:46:ef:3d:24:03:67:d3:d6:f4:f1:b6:6e:19:
                    7d:a0:c4:31:c8:09:32:03:a4:ad:85:ef:b0:82:e8:
                    d3:73:c7:d0:99:9f:84:7f:e8:6f:19:9c:c2:4a:20:
                    57:82:a6:39:04:f3:65:65:4b:2c:80:48:77:88:3b:
                    0a:5f:c4:8e:3e:6c:9f:b1:e1:64:d7:ea:b5:e9:ff:
                    07:5f:10:06:f5:66:3e:26:b8:57:f7:d6:69:de:d1:
                    07:2a:96:15:84:db:db:bd:e8:d3:19:c1:87:84:6a:
                    3e:6e:a7:39:9d:a6:cd:c9:37:60:45:18:93:03:22:
                    73:8e:82:dc:25:57:91:58:d5:37:5c:ba:d0:03:ce:
                    1a:4e:ec:2f:53:5f:e4:39:83:19:7a:85:b5:8d:3c:
                    04:9f:27:cf:52:20:77:af:c4:98:e3:26:b8:61:8b:
                    40:42:4c:e9:d2:1f:e3:7d:45:93:b4:e0:2c:b7:61:
                    30:f5:70:cb:e3:82:2f:56:4f:da:87:41:50:1d:db:
                    67:f3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                9C:C5:17:CF:A0:8D:73:97:E1:25:AA:7C:4B:1B:2B:8C:62:CF:05:07
            X509v3 Authority Key Identifier:
                DirName:/C=ES/ST=Madrid/O=IT-UC3M/CN=Andres Marin
                serial:9F:9B:68:D1:4C:45:09:59

    Signature Algorithm: md5WithRSAEncryption
        13:a9:7f:b0:04:47:a0:4f:7e:6a:50:59:da:fb:73:69:a6:88:
        7a:0d:d0:5d:85:56:19:e7:44:d2:72:59:ce:15:5e:95:9d:52:
        7e:8c:12:6f:0a:51:1a:17:9a:e6:b0:d7:a2:99:76:e9:fc:4a:
        1c:c4:d2:dc:74:a5:73:ad:62:e3:4a:a4:88:af:4f:01:30:4d:
        de:44:be:5d:05:31:1f:d7:63:f0:5e:85:de:d2:71:20:09:ca:
        e4:5e:12:aa:32:16:2d:ae:b6:72:01:3c:09:2c:70:29:ed:ce:
        f3:2b:a9:01:76:7f:bd:da:c4:5a:f0:06:97:65:b2:f2:4a:c6:
        ac:c3:35:0b:24:81:8b:a7:0a:e8:80:77:28:0e:61:cc:2e:75:
        4c:2a:fe:5a:7e:39:4d:7d:16:23:e8:b0:04:54:b0:2a:44:d2:
        fa:15:dd:94:b5:87:8e:6a:c2:95:13:8d:d8:cd:18:f4:14:e4:
        ed:e5:75:c2:b6:a1:81:c9:71:12:e9:1c:fd:36:0d:8e:09:1b:
        42:c7:57:29:f9:e5:06:16:2d:9c:8c:f4:d7:16:80:5f:b1:3c:
        e1:ad:e0:a1:62:a2:32:99:f0:56:26:24:f3:b1:df:3e:f7:1d:
        91:20:03:2b:88:21:b4:d5:9d:1e:ae:8f:95:6c:51:76:56:d5:
        52:87:b9:c5
```

2.2.2.2. Arquitectura de PKI X.509

Una infraestructura de clave pública permite que la tecnología de clave pública se pueda implantar de un modo extenso. Esto se consigue mediante la administración de los certificados de claves públicas de una PKI, lo cual, permite establecer y mantener un entorno de red seguro, de tal manera, que se permite el uso de servicios de cifrado y de firma digital en un gran número de aplicaciones.

En entornos reales, en especial aquellos en los que intervienen una gran variedad de empresas y comunidades de usuarios que trabajan de forma conjunta, surge el problema de cómo estructurar las relaciones entre las entidades de los diversos dominios involucrados.

En la figura 2.1 se muestran los componentes más habituales de una infraestructura de clave pública, que son los siguientes:

- **Autoridad de Certificación (en inglés *Certificate Authority*, CA)**. Es la responsable de emitir y revocar certificados. La Autoridad de Certificación es una entidad de confianza que proporciona legitimidad a la relación existente entre una clave pública y la identidad de un usuario o servicio. Para que una entidad o individuo puedan llevar a a cabo estas tareas, deben acreditarse, lo cual les permitirá ser reconocidos por otras personas, comunidades, empresas o países y que su firma tenga validez. Por ejemplo, una empresa u organización puede distribuir sus propios certificados raíz de confianza a todos los empleados, de manera que ellos puedan usar el sistema de infraestructura de clave pública de la compañía.

- **Autoridad de Registro (en inglés *Registration Authority*, RA)**. Es la responsable de verificar la relación entre la clave pública contenida en un certificado y la identidad de su titular.

- **Repositorios**. Son los sistema o conjunto de sistemas distribuidos que almacenan los certificados y las listas de revocación de certificados (en inglés *Certificate Revocation List* o CRL). En una CLR se incluyen todos aquellos certificados que por algún motivo han dejado de ser válidos antes de la fecha establecida dentro del mismo certificado.

  Los repositorios pueden ser centralizados o estar distribuidos para evitar los cuellos de botella. Estos pueden ser un directorio (LDAP, Lightweight Directory Access Protocol) [17] o una base de datos que puede estar disponible públicamente o ser privada a una organización.

- **Usuarios**. Son aquellos que poseen un par de claves, pública y privada, y un certificado asociado a su clave pública. Utilizan un conjunto de aplicaciones que hacen uso de la tecnología PKI (para validar firmar digitales, cifrar documentos para otros usuarios, etc.)
- **Certificados**. En la sección anterior se explicó con detalle su funcionamiento y estructura. Recordemos que contienen la identidad de un usuario y su clave pública.

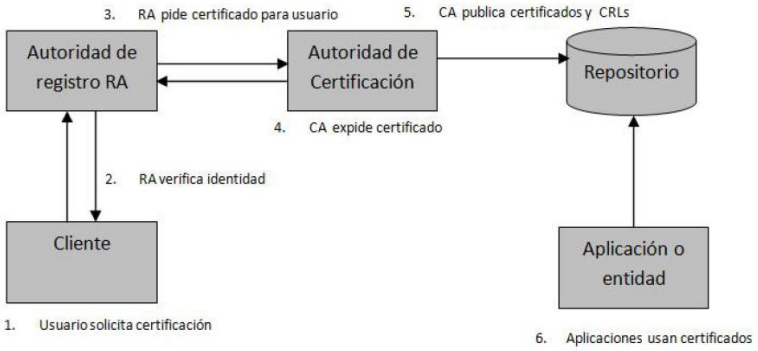

Figura 2.1: Infraestructura PKI

2.2.3. Firma XML

El estándar de firma XML [18] [19](en inglés XML *Signature*) define el formato a utilizar para firmar digitalmente un documento, o partes de un documento XML en el origen, así como para verificar la firma en el destino. De esta forma, XML *Signature* asegura la integridad de partes de documentos XML transportados. Además, proporciona la autenticación de mensajes o servicios de autenticación de firma para datos de cualquier tipo, tanto si se encuentra en el XML que incluye la firma o en cualquier otra parte. Este estándar es aplicable a cualquier contenido digital y éste, representa un sistema que a través de una firma digital permite ofrecer autenticidad de los datos. Con la firma digital se confirma la identidad del emisor, la autenticidad del mensaje y su integridad, además de que los mensajes no serán repudiados.

La novedad que introduce la firma XML es la posibilidad de firmar sólo las partes específicas de un documento XML. Esta propiedad puede resultar interesante, por ejemplo, cuando un documento es creado y firmado por diferentes

personas en distintos instantes de tiempo y cada una ellas firma digitalmente la información que es relevante. Esta flexibilidad permite además, garantizar la integridad de ciertas partes de un documento XML, dejando abierta la posibilidad de que otras partes del documento se puedan modificar. Si se considera el caso de que dicho documento se firmara completamente, el problema que se produciría sería que cualquier modificación desde ese punto en adelante, invalidaría la firma digital original.

Otra de las ventajas que ofrece el estándar de firma XML es que se puede firmar más de un tipo de recurso. Por ejemplo, una sola firma XML podría abarcar los datos de caracteres codificados (HTML), los datos codificados en binario (un archivo JPG), los datos codificados en XML, y una sección específica de un archivo XML. En el estándar se definen también etiquetas por ejemplo, para definir el recurso a firmar, para llevar a cabo la gestión del recurso que se va a firmar, para la representación de la firma digital, etc. En el siguiente ejemplo, se muestra un documento XML firmado:

Documento XML firmado

```
<Signature Id="EjemploXMLSignature"
 xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
     <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
     <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
     <Reference
        URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
      </Reference>
   </SignedInfo>
   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
   <KeyInfo> <KeyValue> <DSAKeyValue>
           <p>...</p><q>...</q><g>...</g><y>...</y>
         </DSAKeyValue>
      </KeyValue>
   </KeyInfo>
```

Se pueden identificar las distintas partes de la que consta un documento XML firmado, definidas por ciertas etiquetas. La etiqueta `Signature` encapsula la firma digital. Ésta consta de tres elementos: `SignedInfo`, `SignatureValue` y `KeyInfo`. El elemento `SignedInfo` contiene información acerca de cómo crear y validar la firma. Además, en este elemento se pueden apreciar dos algoritmos. El primero está encapsulado dentro de la etiqueta `<CanonicalizationMethod>` y es el algo-

ritmo que transforma el elemento `SignedInfo` antes de hacer la firma digital. El segundo algoritmo se encapsula dentro de la etiqueta `<SignatureMethod>` y es el algoritmo que se utiliza para realizar el cálculo del valor de la firma digital. En el elemento `SignedInfo` se incluyen las referencias a los objetos que se van a firmar, por medio de la etiqueta `<Reference>`, la cual a su vez, contiene los elementos `<DigestMethod>` y `<DigestValue>`. Para llevar a cabo la validación de una firma, es necesario realizar dos pasos: la validación de la firma y la validación de los resultados de las referencias.

Otro elemento a resaltar, es el `<SignatureValue>`, que contiene el resultado de la firma digital que se ha aplicado sobre un elemento de tipo `SignedInfo`. El resultado de esta firma está codificado y contiene un atributo que es único con el que se identificará la firma en procesos posteriores de validación.

En el ejemplo, se pueden observar otros elementos como `<KeyInfo>` y `<KeyValue>`, cuyo carácter es opcional e indican la clave que debe utilizarse para validar la firma.

El receptor de este documento firmado debe ser capaz de verificar la firma del certificado digital, así como validar el certificado que corresponde con la clave privada del emisor del documento para generar la firma digital. La validación de una firma XML requiere que los datos que se firmaron sean accesibles. El documento XML se firma por lo general indicando la ubicación de la firma en el objeto original.

2.3. Gestión de Identidad Federada

El objetivo principal de la gestión de identidad federada es permitir que identidades y atributos de entidades sean compartidos a través de distintos dominios de confianza según las políticas establecidas. Estas políticas especifican aspectos tales como, formatos, opciones, requisitos de privacidad y confianza. Los sistemas de gestión de identidad que se analizan a lo largo de esta sección, necesitan una infraestructura de autenticación, como por ejemplo, PKI o Kerberos. También requieren una infraestructura de gestión de privilegios para ofrecer servicios de autorización.

Las distintas especificaciones de gestión de identidad, surgen como respuesta a la necesidad de separar las tareas de povisión de servicios de aquéllas dedicadas a la gestión de identidad , de tal manera, que se pueda simplificar la gestión de identificadores, credenciales, atributos, etc., y con ello, se pueda ofrecer al usuario una experiencia satisfactoria cuando interactúe con distintos dominios administrativos.

2.3.1. Security Assertion Markup Language (SAML)

SAML es un marco de trabajo que permite expresar asertos acerca de la identidad, los atributos y las autorizaciones de un sujeto con el objetivo de facilitar las relaciones entre distintas empresas, así como las relaciones de éstas con sus usuarios. Este marco de trabajo permite a las compañías crear identidades federadas, lo cual les facilita las tareas de gestión de perfiles, autenticación y autorización de usuarios. El caso típico de uso es el de *Single-Sign-On* (SSO), que permite a los usuarios acceder a diversos sitios en la federación con una única autenticación.

Las especificaciones de SAML se basan en el estándar XML, y definen esquemas y formatos para los distintos elementos y mensajes. Cabe destacar que en la actualidad, es el único estándar abierto y es la base de otras especificaciones de federación de identidad como las de Liberty Alliance, que se presenta en la siguiente sección, y propuso una extensión de SAMLv1.0 en su especificación ID-FF 1.1 en Enero de 2003. En Noviembre de ese mismo año, Liberty lanza una nueva especificación, ID-FF 1.2 [3] [4], basada en SAMLv1.1. Finalmente, en marzo de 2005, SAML v2.0 es anunciado como estándar de OASIS y representa la convergencia de Liberty ID-FF y otras iniciativas de gestión de identidad, como Shibboleth de Internet2 o Web Services Security (WSS)de OASIS.

No obstante, aunque SAML se estandarizó en 2005, todavía se continúan realizando rectificaciones y nuevas especificaciones (rectificación de Agosto de 2007 "*Approved Errata for SAML V2.0*", *draft* de revisión técnica en Marzo de 2008, etc.).

El estándar SAML define cuatro elementos fundamentales:

- **Asertos (en inglés, *Assertions*)**. Los asertos definen las afirmaciones de seguridad de una entidad dentro de un sistema. Estas afirmaciones pueden ser de tres tipos: asertos de autenticación, de atributos y de decisiones de autorización. Los asertos de autenticación indican que el usuario ha sido autenticado por alguna aplicación, mientras que los asertos de atributo son más específicos e incluyen información sobre los atributos del usuario (por ejemplo que un usuario pertenece a un determinado grupo con más privilegios). Por último, encontramos los asertos de decisión de autorización, más específicos todavía, que incluyen directamente los privilegios que posee un usuario (por ejemplo que un usuario tenga permisos para comprar cierto producto). Estos asertos y otros componentes del estándar se basan en XML, lo que permite que puedan ser utilizados en otros contextos. A continuación se muestra un ejemplo de un aserto de autenticación y de atributos:

Ejemplo de aserto de autenticación y atributo

```
<saml:Assertion Version="2.0" IssueInstant="2004-12-05T09:22:05Z">
  <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
   <ds:Signature
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
    </ds:Signature>
     <saml:Subject>
       <saml:NameID
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:
        transient"> 3f7b3dcf-1674-4ecd-92c8-1544f346baf8
       </saml:NameID>
      </saml:Subject>

<saml:AuthnStatement AuthnInstant="2004-12-05T09:22:00Z"
 <saml:AuthnContext>
  <saml:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
     </saml:AuthnContextClassRef>
  </saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
  <saml:Attribute
     xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:
     X500" x500:Encoding="LDAP"
     NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
     Name="urn:oid:1.3.6.1.4.1.5923.1
     FriendlyName="eduPersonAffiliation">
    <saml:AttributeValue
       xsi:type="xs:string">member</saml:AttributeValue>
      <saml:AttributeValue
             xsi:type="xs:string">staff</saml:AttributeValue>
     </saml:Attribute>
   </saml:AttributeStatement>
</saml:Assertion>
```

- **Protocolos (en inglés, *Protocols*)**.

  SAML define un conjunto de protocolos de solicitud/respuesta que describe un mecanismo de intercambio automático de asertos. Se definen seis protocolos, que se comentan brevemente a continuación:

  - **Protocolo de petición de autenticación (en inglés *Authentication Request Protocol*)**.

    Este protocolo se utiliza por ejemplo, cuando un proveedor de servicios debe autenticar a un usuario. La respuesta a este mensaje de petición de autenticación debe contener por lo menos un aserto de autenticación

y puede contener además, uno o varios asertos de atributo y de decisión de autorización.

- **Protocolo de registro único de salida (en inglés, *Protocol Single Logout*).**

  Este protocolo especifica la manera de realizar un *logout* simultáneo de todas las sesiones asociadas a un determinado usuario. El *logout* puede ser iniciado directamente por el usuario, o iniciado por un IdP o SP debido por ejemplo, a la expiración de la sesión por tiempo o por decisión de un administrador.

- **Protocolo *Assertion Query and Request*.**

  Define un conjunto de consultas por la cuales se pueden obtener asertos SAML.

- **Protocolo de gestión de identificadores (en inglés *Name Identifier Management Protocol*)**

  Este protocolo define mecanismos para modificar el valor o formato del identificador utilizado para referirse al usuario. Además, especifica la forma de terminar una asociación de la identidad del usuario entre un IdP y un SP.

- **Protocolo de mapeo de identificadores (en inglés *Name Identifier Mapping Protocol*)**

  Este protocolo define un mecanismo que permite mapear un identificador de usuario usado entre un IdP y un SP de forma de obtener el utilizado entre el IdP y otro SP.

- **Protocolo *Artifact Resolution***

  Este protocolo permite que se puedan transportar los asertos por referencia (a esta referencia se le denomina *Artifact*) en vez de valor. El protocolo especifica cómo obtener el aserto dada la referencia.

- ***Bindings*.**

  Los asertos y los protocolos están definidos como un esquema XML. Para que se puedan utilizar en la práctica, es necesario realizar un mapeo a los distintos protocolos de transporte utilizados (HTTP-GET, HTTP-POST, SOAP, etc.). Se definen seis maneras distintas de transportar los mensajes de los protocolos en otros protocolos estándar.

- **Perfiles (en inglés, *Profiles*).**

  Los protocolos y los *bindings* por sí solos no permiten la interoperabilidad de SAML. Los perfiles definen la secuencia específica de mensajes y los

bindings requeridos en cada caso, para completar cada uno de los casos de uso definidos en el estándar. Por cada combinación de caso de uso y *binding* se pueden obtener variaciones de un mismo perfil. En el estándar se definen ocho asociaciones distintas de protocolos y transportes específicos a un caso de uso/aplicación (*Web SSO profile*, *Attribute profile*, etc.).

Existen diversas implementaciones de comerciales y de código abierto de SAML en distintos lenguajes de programación. Por ejemplo, existen implementaciones [27] en C (ZXID), C++/Java (OpenSAML), Java (OpenSSO), PHP (simpleSAMLphp), Python, etc. SAML también está soportado en proyectos como Shibboleth, desarrollado por Internet2 y que se trata de una implementación de perfiles construida sobre OpenSAML. WS-Security e ID-FF también lo soportan. En los siguientes capítulos de este proyecto se analizarán más en profundidad algunas de estas implementaciones.

2.3.1.1. Arquitectura de gestión de identidad en SAML

En este apartado se expondrán brevemente las principales entidades que pueden aparecer en un sistema de gestión de identidad federada basado en SAML, así como los distintos papeles que pueden desempeñar dichas entidades. Antes de continuar, cabe aclarar que a lo largo de esta sección se utilizará el término "entidad SAML" para referirse a un elemento activo de un sistema de red (p.ej. un proceso, persona o grupo de personas) que incorpora un conjunto de funcionalidades determinadas.

En este tipo de sistemas suelen aparecen como mínimo dos entidades con los roles de Proveedor de Servicios (SP, *Service Provider*) y Proveedor de Identidad (IdP, *Identity Provider*) respectivamente, y un Usuario Final que interactúa con dichos proveedores por medio de un agente de usuario.

A continuación se explican brevemente las distintas funciones llevadas a cabo por cada una de estas entidades:

- **Proveedor de Servicios (SP)**.

  Se trata de la entidad encargada de proporcionar un determinado servicio a aquellos usuarios o entidades SAML que confían en el IdP para llevar a cabo las tareas de identificación. Este papel lo desempeñan un tipo de entidades SAML denominadas *Relying Party* (RP). Las decisiones tomadas por un SP se basan en la información que le ha proporcionado otra entidad del sistema acerca de un determinado usuario.

- **Proveedor de Identidad (IdP)**.

Se trata de la entidad encargada de emitir, mantener y gestionar la información de identidad relativa a los usuarios en una federación. Este papel lo desempeñan un tipo de entidades SAML denominadas *Asserting Party* (AP). El IdP es el que posee la infraestructura de autenticación e implementa la funcionalidad para llevar a cabo el almacenamiento de credenciales, el borrado y la recuperación de las mismas por parte del usuario, etc. Además, los IdPs emiten asertos SAML para que puedan ser utilizados por los SPs a la hora de tomar decisiones sobre un Usuario Final.

- **Usuario Final**

  Se trata de aquel que interacciona con los servicios proporcionados por el SP y también, es el sujeto de los asertos emitidos por el IdP. Generalmente, el Usuario Final accede a los servicios por medio de un agente de usuario (por ejemplo, el navegador del teléfono móvil para acceder a un servicio web).

2.3.1.2. Seguridad en SAML

En SAML se definen unas declaraciones de contextos de autenticación y una serie de clases para ello. El objetivo es poder dar información adicional acerca del nivel de confianza en la autenticación, que se deriva directamente de las tecnologías, protocolos y procesos que se han utilizado para dicha autenticación. En los esquemas que se han definido hasta el momento, están considerados prácticamente todos los esquemas de autenticación que se utilizan actualmente, y además cuenta con la ventaja de que el mecanismo es extensible. Algunos de estos contextos son: *main schema, common schema types, IP, IP password, Kerberos, mobile one-factor contract, mobile one-factor unregistered, mobile two-factor contract, mobile two-factor unregistered, personal telephony, PGP, password-protected transport, password, smartcard, smartcard PKI, software PKI, SPKI, SSL certificate, authenticated telephony, time sync token, X.509, XML Signature,etc.*

SAML no define cifradores, códigos de integridad o firmas digitales para autenticar o garantizar la integridad o privacidad de sus mensajes. En SAML se requiere la preexistencia de una relación de confianza previa entre la parte que solicita un aserto (RP) y el que lo emite (AP). Esta relación de confianza previa, normalmente se obtiene por medio del uso de una PKI. En los distintos perfiles, se recomienda:

- utilizar HTTP [23] sobre SSL/TLS (HTTPS), en aquel caso en el que es necesario garantizar la integridad y confidencialidad de un mensaje.

- utilizar autenticación de ambas partes cliente/servidor de SSL/TLS, en el caso de que se tenga que enviar una solicitud de un aserto.
- utilizar XML, en el caso de que se requiera devolver un aserto, dado que el mensaje de respuesta debe ir firmado.

2.3.1.3. Federación de identidad en SAML

En SAML se definen múltiples casos de uso [2], a continuación se exponen algunos de los principales escenarios en federación de identidad:

1. **Federación enlazando cuentas fuera de banda (en inglés, *Federation via Out-of-Band Account Linking*)**.

   Consiste en el establecimiento de identidades federadas de usuarios y la asociación de dichas identidades a identidades de usuarios locales.

2. **Federación por atributos de identidad (en inglés, *Federation via Identity Attributes*)**.

   En este caso de uso, el IdP define unos atributos que se utilizan a la hora de enlazar la cuenta utilizada por un usuario en el SP.

3. **Federación utilizando pseudónimos persistentes (en inglés, *Federation via Persistent Pseudonym Identifiers*)**.

   Consiste en que un proveedor de identidad federa la identidad del usuario local con la identidad del usuario en el proveedor de servicios utilizando un identificador de nombre persistente de SAML.

4. **Federación utilizando pseudónimos temporales (en inglés, *Federation via Transient Pseudonym Identifiers)*.**

   En este caso de uso se utilizan identificadores temporales entre el IdP y el SP, que únicamente son válidos durante la sesión (SSO).

5. **Terminación de la federación (en inglés, *Federation Termination*)**.

   En este caso se finaliza una federación existente (sesión e identificadores temporales en caso de haberlos).

En la figura 2.2, se muestra un ejemplo del caso 1, que permite una sesión SSO a un usuario conocido como John en el proveedor de identidad `airline.example.com` y en el proveedor de servicios `cars.example.co.uk`, mediante los siguientes procesos:

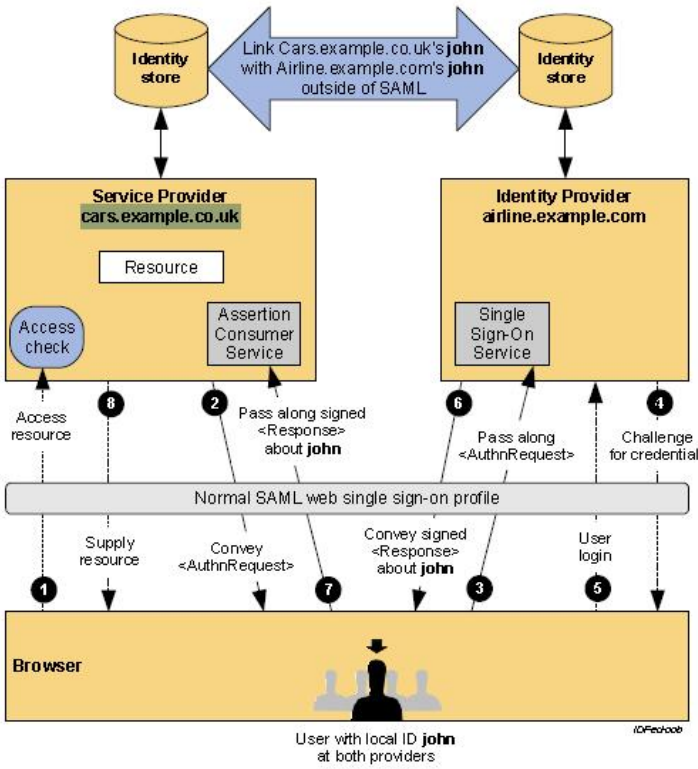

Figura 2.2: Ejemplo de federación de identidad enlazando cuentas fuera de banda

1. John desea iniciar sesión en el SP, accede al IdP para autenticarse, y el IdP le envía un reto de autenticación (de acuerdo con los contextos de autenticación utilizados por el IdP: contraseña, certificado, etc.).

2. John se autentica con éxito frente al IdP (por ejemplo enviando sus credenciales).

3. John indica al IdP que quiere acceder a un recurso del SP (por ejemplo mediante una opción del menú del IdP).

4. El IdP envía un formulario HTML a John en el que se incluye la respuesta SAML (el aserto SAML firmada por el IdP sobre Bob).

5. John envía dicha respuesta SAML al SP.

6. El servicio consumidor de asertos del SP valida la firma del aserto, crea una nueva sesión local (basada en la cuenta local de John) y envía a John una *cookie* que identifica la nueva sesión. Si conoce el recurso que quería acceder

John, puede acompañar la *cookie* de una redirección para acceder al recurso solicitado.

7. John solicita el recurso deseado (bien directamente o siguiendo la redirección) y el SP comprueba si John está autorizado para acceder en cuyo caso permite el acceso.

2.3.2. Liberty Alliance

La iniciativa Liberty Alliance surgió en Septiembre de 2001, tras la unión de un consorcio de empresas, proveedores e instituciones con un interés común: proporcionar estándares para gestión de identidades federadas.

El modelo de federación de Liberty, proporciona garantías de privacidad y seguridad, además de un mecanismo abierto y estándar de registro único (el caso comentado en la sección 2.3.1 de *Single Sign-On*).

El desarrollo del conjunto de estándares y recomendaciones se ha estructurado en tres fases:

- Fase 1. *Identity Federation Framework* (ID-FF).
- Fase 2. *Identity Web Services Framework* (ID-WSF).
- Fase 3. *Identity Services Interface Specifications* (ID-SIS).

ID-FF propone el uso de identidad de red federada para solucionar los problemas de identidad de red. ID-WSF se basa en ID-FF para proporcionar un marco para los servicios web basados en identidad federada. Proporciona los mecanismos necesarios para compartir atributos basados en permisos, descubrimiento de perfiles y servicio de interacción con el usuario. La especificación incluye el soporte de SAML y permite implementar servicios web sobre un entorno estándar que garantiza la seguridad extremo a extremo en las invocaciones y el envío de mensaje.

ID-SIS, por su parte, utiliza tanto ID-FF como ID-WSF para proporcionar servicios de red. ID-SIS especifica interfaces de servicios web de alto nivel que soportan casos de uso particulares como perfiles, localización geográfica, presencia, etc.

2.3.2.1. Identity Federation Framework (ID-FF)

Identity Federation Framework fue la primera especificación Liberty y define los roles de las entidades participantes en un entorno de identidad federada, así como los protocolos necesarios para llevar a cabo las operaciones de federación de identidades, registro único de entrada (SSO), utilización de pseudónimos globales y registro único de salida ("Single Logout").

El concepto de círculo de confianza (en inglés, *Circle of Trust*), aparece como una de las claves de esta especificación, y se define como, un conjunto de entidades (compañías, departamentos, administraciones públicas, etc.) que han firmado un acuerdo de negocio para suministrar una serie de servicios a sus usuarios comunes, basado en una relación de confianza entre ellos, en principio a efectos de la autenticación de dichos usuarios.

En las especificaciones que componen el módulo ID-FF, se refleja la descripción de la arquitectura[4], protocolos abstractos y los XSD [3] utilizados, las líneas de implementación recomendadas, la especificación de requisitos obligatorios y opcionales, etc. Un ejemplo de implementación en código abierto, es Lasso (*Liberty Alliance Single Sign-On*)[24], que se trata de una librería escrita en C y que ofrece "*bindings*" para Java, Perl, Python y PHP. Esta librería soporta Liberty ID-FF 1.2, ID-WSF y SAMLv2 y ha tenido una especial relevancia en el desarrollo de este proyecto.

Arquitectura En una implementación de Liberty ID-FF, se reutilizan y adaptan las definiciones de entidades y roles de SAML que se vieron en el apartado 2.3.1, de tal manera, que se ven involucrados los siguientes elementos:

- **Principal (o entidad final)**: Puede ser un usuario individual, un grupo de individuos, una corporación o un componente de la arquitectura Liberty. También puede tener definida una identidad local con más de un proveedor y tiene la opción de federar las identidades.
- **El Proveedor de Servicios (SP)**: Se trata del equivalente a la "*relying party*" en SAML.
- **Proveedor de Identidad (IdP)**: Se trata del equivalente a la "*asserting party*" en SAML y posee la infraestructura necesaria para ofrecer servicios de autenticación.

Para ofrecer federación de identidad, los proveedores de servicios y los proveedores de identidad, deben juntarse para formar un dominio de autenticación,

que debe contener por lo menos un proveedor de identidad y dos proveedores de servicios. Las organizaciones de un dominio de autenticación, deben acuerdos que definan sus relaciones dentro de un círculo de confianza.

2.3.3. OpenID

OpenID es un marco para la gestión de la identidad digital, que surgió en 2005 y se caracteriza por ser descentralizado, abierto y centrado en el usuario. Su funcionamiento se apoya en tecnologías de Internet ya existentes y ampliamente utilizadas, envía los mensajes sobre el protocolo HTTP, y para reforzar la seguridad, recomienda el uso de Diffie-Hellman [26] y SSL.

El funcionamiento principal de OpenID se basa en que una identidad puede venir dada bien por una URI o bien, por un XRI, y puede ser verificada por un servidor que soporte el protocolo. Este protocolo se encarga únicamente de proporcionar una forma de demostrar que alguien es el propietario de cierto identificador. Además, OpenID soporta SSO, lo cual ofrece la ventaja que se ha comentado en ocasiones anteriores a lo largo de este capítulo, de reducir la cantidad de credenciales de identidad que deben ser mantenidas por un usuario para acceder a distintos servicios en Internet. Otro de los objetivos de esta iniciativa, es que los sitios Web puedan aceptar credenciales de manera sencilla y poco costosa.

El hecho de que OpenID sea descentralizado significa que cualquiera puede actuar como IdP sin necesidad de registrarse ni de obtener permiso de alguna organización o autoridad central. De esta característica, se deriva la ventaja de que cualquier usuario es libre para elegir un proveedor en el que confíe y que satisfaga sus necesidades. También ofrece servicios de delegación, lo que supone una gran ventaja para el usuario, al no estar éste ligado a un proveedor concreto sino que puede elegir y cambiar de proveedor siempre que lo desee y sin necesidad de cambiar de identificador.

Otros de los aspectos que se comentaron al principio de esta sección, es el carácter abierto de OpenID, es decir, su especificación es pública y está basada en estándares conocidos de Internet y además, está centrado en el usuario. Esta última propiedad, permite al usuario controlar su información y decidir qué datos personales deben enviarse cada vez que acceda a un servicio.

Desde su aparición, ha ido incrementando considerablemente el número de IdPs y de páginas web que soportan acceso OpenID. Se pueden encontrar implementaciones de código abierto [28] en Java (OpenID4Java, WSO2 OpenID Library, JOpenID, etc.), en C++ (libopkele), etc. Además, grandes compañías como AOL, Yahoo, Microsoft, VeriSign, Google o IBM han adoptado este sistema. En

España existen varios proveedores de identidad como: openid.es , openid.blogs.es , miID.es y Movistar OpenID , que es el primer operador móvil en lanzar un IdP OpenID.

Funcionamiento y vulnerabilidades de OpenID

En un sistema de identidad OpenID existen tres elementos fundamentales: un usuario con un navegador Web, un Consumidor, que sería el sitio Web que ofrece el servicio al que desea acceder el usuario, y un Proveedor de Identidad OpenID (IdP), que se encarga de verificar que un usuario posee un determinado identificador URI o XRI. Además, el proveedor puede disponer de ciertos elementos extra de información de identidad acerca del usuario.

En el flujo de mensajes intercambiado en un diálogo OpenID, se llevan a cabo una serie de acciones que se describen a continuación :

1. Un usuario final introduce su identificador en el sitio Web de un consumidor utilizando el navegador.

2. Se normaliza el identificador presentado por el usuario y el consumidor lleva a cabo el descubrimiento de la localización del IdP OpenID utilizado por el usuario para ser autenticado.

3. Opcionalmente, el consumidor puede establecer una asociación con el proveedor. En este caso, ambas partes intercambian un secreto compartido siguiendo la especificación de Diffie-Hellman.

4. El consumidor redirige el navegador al IdP por medio de un mensaje de petición de autenticación.

5. El IdP pide al usuario que se autentique y redirige el navegador de nuevo al consumidor por medio de un mensaje de respuesta. Además, el consumidor puede solicitar el envío de cierta información personal del usuario, pero esta sólo será enviada bajo su consentimiento.

6. El consumidor verifica la respuesta recibida del IdP y, en caso de que la autenticación sea positiva, el usuario tendrá acceso al servicio.

En el caso de que el usuario ya se haya autenticado previamente, el proveedor envía una confirmación al consumidor de manera transparente al usuario sin necesidad de interaccionar con él, proporcionando así la funcionalidad de SSO.

A pesar de que OpenID ofrece multitud de ventajas en las tareas de gestión de identidad, también presenta algunas vulnerabilidades, que se comentan brevemente a continuación. No existe un modelo de confianza y ni de seguridad, sino que el protocolo se basa en el principio que *confía y acepta a todos los que llegan*. A pesar de que para cierto tipo de servicios esto puede ser suficiente, este mecanismo convierte a OpenID en un sistema con muchas vulnerabilidades para otras aplicaciones que requieren requisitos más estrictos de seguridad, debido a que define un formato de mensajes tan sencillo que hace que siempre se revele la identidad del consumidor al IdP. Otro aspecto que tampoco se trata en profundidad en la especificación, es el mecanismo de autenticación que se utiliza para verificar que un usuario posee una identidad. Por lo tanto, la seguridad de una conexión depende de la confianza que tenga el consumidor en el IdP.

Estas vulnerabilidades hacen que el protocolo sea bastante sensible a ataques de *phishing*, debido a que gran parte de la comunicación se basa en redirecciones del navegador. Para solucionar este problema, se recomienda la utilización de *plug-ins* en el navegador que verifiquen la autenticidad de la URL del servidor de identidad, el uso del protocolo SSL en todas las interacciones de la comunicación y además, se ha propuesto una extensión, denominada *Provider Authentication Policy Extension* (PAPE)[29], que proporciona un mecanismo para que el RP pueda solicitar al IdP que aplique una determinada política de autenticación para comprobar la identidad del usuario.

2.3.4. WS-Federation

WS-Federation [30] es una especificación de federación de identidad desarrollada por BEA Systems, BMC Software, CA Inc., IBM, Layer 7 Technologies, Microsoft, Novell y VeriSign, que se encuentra en proceso de estandarización dentro de OASIS (WS-FED, documentos de Mayo de 2009 "Web Services Federation Language (WS-Federation) Version 1.2" [9]). Esta especificación forma parte del marco de seguridad de Servicios Web WS-* y se describe cómo utilizar WS-Trust, WS-Security y WSPolicy, así como la forma de proporcionar seguridad entre dominios federación. Se definen mecanismos que permiten a distintos reinos de seguridad negociar la información sobre las identidades, atributos y asertos de autenticación y autorización para controlar el acceso a los recursos.

La arquitectura de WS-Federation separa mecanismos de confianza, formatos de tokens de seguridad, y el protocolo para obtener dichos *tokens*.

Funcionamiento de WS-Federation

En servicios web, la federación permite negociar las identidades y atributos desde los emisores de *tokens* de seguridad e identidad hasta los servicios y otras *relying parties* sin necesidad de una intervención por parte del usuario, a menos que se especifique en las políticas correspondientes. Para ello, es necesario realizar una configuración inicial de los distintos sistemas implicados, lo cual supone llevar a cabo un intercambio de metadatos de la federación, que describan la información acerca de los servicios federados, de las políticas que describen los requisitos comunes de comunicación, y de la negociación de confianza y *tokens*.

Para establecer un contexto de federación para una entidad, su identidad debe ser universalmente aceptada o negociada en una identidad fiable para cada reino de confianza dentro del contexto de la federación. Este último caso, requiere realizar un proceso conocido como mapeo de identidades, que consiste en la conversión de una identidad digital en un reino a otra identidad digital válida en otro reino por una parte que confía en el reino origen y tiene los derechos para hacer declaraciones en el reino destino o en las cuales éste confía. Las entidades encargadas de llevar a cabo este mapeo de identidades son un proveedor de identidad o un STS (*Security Token Service*), obteniendo los *tokens* para un servicio o intercambiándolos con un IdP del servicio.

Existen algunas implementaciones de código abierto de esta especificación, como por ejemplo: OpenWSFed [31], que ha sido desarrollada utilizando componentes del marco de trabajo OpenSAML 2.0. Sin embargo, el código de esta librería se encuentra todavía en proceso de desarrollo y experimentación.

2.4. Protocolo de Acceso Ligero a Directorios (LDAP)

2.4.1. Introducción

En esta sección se exponen algunas nociones básicas acerca del protocolo LDAP [32], que permitirán al lector poder comprender mejor algunos aspectos de la arquitectura del sistema de gestión de identidad desarrollado a lo largo de este proyecto. LDAP es un protocolo de acceso a directorios sobre TCP/IP. Un servidor LDAP se utiliza para procesar consultas y actualizaciones de un directorio de información LDAP. Este último, se trata de un tipo de base de datos no relacional, que está diseñado para realizar lecturas de datos de forma muy eficiente.

Generalmente, los datos de un directorio LDAP siguen un modelo de organización de información en árbol según [14]. Una de las principales ventajas que ofrece LDAP frente a otras tecnologías como SQL, por ejemplo, es que aquellos atributos que por algún motivo no se lleguen a utilizar, no ocupan espacio de almacenamiento en los objetos almacenados.

El protocolo LDAP define el contenido de los mensajes intercambiados entre un cliente LDAP y un servidor LDAP. Estos mensajes especifican las operaciones solicitadas por el cliente (insertar, borrar, modificar, etc.), las respuestas del servidor y el formato de los datos transportados en los mensajes.

2.4.2. Funcionamiento y administración de LDAP

Como se ha comentado, el servicio de directorio LDAP se basa en un modelo cliente-servidor. El funcionamiento básico de este protocolo consiste en que uno o más servidores contienen los datos que conforman el árbol de directorio , el cliente LDAP se conecta con el servidor LDAP y realiza una consulta. El servidor por su parte, proporciona al cliente una respuesta a su petición o una indicación de dónde puede obtener la información deseada. Independientemente del servidor LDAP al que se conecte el cliente, éste siempre observará la misma vista del directorio.

A continuación se exponen algunos aspectos relevantes en lo que se refiere a la administración de LDAP, tales como la definición de términos en la estructura de árbol: creación de entradas, atributos LDAP y el formato de ficheros en LDAP.

- **Entradas LDAP**. El modelo de información de LDAP se basa en entradas. Una entrada se define como una colección de atributos que tienen un *Nombre Distintivo* (DN) único y global . El DN se utiliza para referirse a una entrada de forma unívoca. Cada atributo de una entrada determinada posee un tipo y uno o más valores. Algunos ejemplos de tipos definidos son "cn" para referirse al nombre de pila (en inglés, *common name*), o "mail" para especificar una dirección de correo. La sintaxis de los atributos depende del tipo de atributo.

- **Atributos LDAP**. Los datos del directorio se representan mediante pares de atributo y su valor. Existen dos tipos de atributos: atributos requeridos y atributos opcionales. Los primeros, deben estar presentes obligatoriamente en las entradas que utilicen en la clase de objetos. Los atributos opcionales, en cambio, pueden no aparecer en dichas entradas. Por ejemplo, en la clase de objeto *person*, los atributos *cn* y *sn* son obligatorios; mientras que los atributos *description*, *telephoneNumber* y *userpassword* son opcionales.

Cada atributo tiene la definición de sintaxis que le corresponde. La definición de sintaxis describe el tipo de información que proporciona ese atributo. Las especificaciones relativas a la definición de atributos se recogen en [33].

- **Objetos LDAP**. Una clase de objetos representa la colección de atributos que pueden utilizarse para definir una cierta entrada. En el estándar se especifican cuatro tipos básicos para las clases de objetos, que se comentan brevemente a continuación:

  1. Grupos en el directorio.
  2. Emplazamientos, por ejemplo el nombre del país y su descripción.
  3. Organizaciones que se encuentran en el directorio.
  4. Personas que se encuentran en el directorio.

  Otro aspecto a destacar, es que una entrada determinada puede pertenecer a más de una clase de objetos. Por ejemplo, la entrada utilizada para representar personas se define mediante la clase de objetos *person*, pero también puede definirse mediante atributos en las clases de objetos *inetOrgPerson*, *groupOfNames* y *organization*. Además, LDAP también ofrece la posibilidad de definir esquemas de clases de objetos propios, que contengan atributos que se puedan ajustar mejor a las necesidades de una determinada aplicación. La estructura de clases de objetos de un servidor LDAP determina la lista total de atributos requeridos y permitidos para una entrada concreta.

- **LDIF (*LDAP Data Interchange Format*)**. Se trata de un tipo de fichero (normalmente un fichero ASCII) que especifica el formato de intercambio de LDAP. Un fichero LDIF almacena información en jerarquías de entradas. Generalmente, se utiliza para importar y exportar información de directorios entre servidores de directorios basados en LDAP, o para describir una serie de cambios que se deben aplicar a dicho directorio. A continuación, se muestra un ejemplo de una entrada para describir una cuenta de usuario en un servidor:

Ejemplo de formato LDIF para una cuenta de usuario

```
dn: uid=rsanchez,ou=People,dc=idp.gast.it.uc3m,dc=es
uid: rsanchez
cn: Rosa Sanchez
objectclass: account
objectclass: posixAccount
objectclass: top
loginshell: /bin/bash
uidnumber: 512
gidnumber: 300
homedirectory: /home/rsanchez
gecos: Rosa Sanchez,,,,
userpassword: {crypt}LPnaOoUYN57Netaac
```

Como se puede observar, el tipo de objeto utilizado para crear esta cuenta es *posixAccount*, que posee como atributos obligatorios cn y *uid*, los cuales proporcionan el nombre del usuario y un identificador. Además, este tipo de objeto dispone de atributos opcionales que permiten ofrecer más información acerca de la cuenta creada, como por ejemplo, la contraseña de acceso, la ubicación del directorio de dicha cuenta, etc.

2.4.3. Seguridad y control de acceso

El protocolo LDAP cuenta con un complejo nivel de instancias de control de acceso. Además, presenta la ventaja de que el acceso puede ser controlado desde el lado del servidor, lo cual resulta más seguro que realizar dicho control desde el lado del cliente. El sistema de control de acceso ofrecido permite indicar a qué recurso se da acceso, quién(es) tiene(n) acceso a ese recurso, así como qué tipo de acceso (lectura,escritura,etc.) se tiene. Esta propiedad resulta de gran utilidad, pues habrá casos en los que cierta parte de la información almacenada en el directorio pueda ser leída por todos usuarios y sin embargo; en otros casos, esta información sólo pueda ser accesible o modificable por un usuario con permisos de administrador.

Para acceder a un servicio LDAP, el cliente LDAP primero debe autenticarse en dicho servicio. Si el cliente se autentica correctamente y realiza una solicitud, el servidor comprueba si éste dispone de los permisos necesarios para realizar su petición. El estándar LDAP especifica formas para que los clientes pueden realizar la autenticación LDAP a servidores LDAP en la RFC 2251 [17] y la RFC 2829 [34]. Éstas se tratan en general en las secciones de autenticación LDAP y mecanismos de autenticación. Algunos de los mecanismos de autenticación que soporta LDAPv3 son: autenticación anónima, autenticación simple (contraseña

en texto claro), Kerberos y SASL (*Simple Authentication and Security Layer*)[35]. Éste último, es un estándar que separa los mecanismos de autenticación de los protocolos de la aplicación, de tal forma que, cualquier protocolo de aplicación que use SASL, puede utilizar cualquier mecanismo de autenticación soportado por SASL. Mediante SASL sólo se maneja la autenticación, se requieren otros mecanismos, como por ejemplo TLS, para cifrar el contenido que se transfiere. Un mecanismo SASL se modela como una sucesión de retos y respuestas.

Otro aspecto de seguridad que también permite LDAP, es la transmisión de solicitudes y respuestas a través de SSL entre el cliente y el servidor. Esta opción puede resultar útil por ejemplo, para enviar y recibir los atributos que contienen información confidencial (contraseñas, claves, etc.).

2.5. Lenguajes de programación utilizados

En el desarrollo de este proyecto se han empleado diversos lenguajes de programación, en especial aquellos orientados al desarrollo de servicios web. A lo largo de esta sección se expondrán los aspectos principales y las características de dichos lenguajes.

2.5.1. Python

Python [36] es un lenguaje de programación de alto nivel que en los últimos años ha ganado una gran popularidad entre los desarrolladores web. Este lenguaje se caracteriza principalmente por definir una sintaxis muy sencilla y legible, a la vez que consigue tener una "elasticidad" perfecta para la web.

Es comparado habitualmente con otros lenguajes como TCL, Perl, Java y Ruby. En la actualidad, Python se desarrolla como un producto de código abierto, administrado por Python *Software Foundation* [1]. La última versión estable de este lenguaje es la 3.1.1.

Existen versiones disponibles de Python en múltiples platataformas (Linux, Windows, Macintosh, Solaris, etc). Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Además, incluye módulos que proporcionan E/S de ficheros, llamadas al sistema, *sockets* y e interfaces gráficas con el usuario.

[1] http://www.python.org/psf/

Otras características interesantes, son sus tipos completamente dinámicos, que junto a la administración automática de memoria, permite realizar programas sin necesidad de realizar tareas de tan bajo nivel.

Por otra parte, soporta el modelo orientado a objetos, al igual que otros lenguajes conocidos como Java, C++, etc; pero también permite seguir un modelo de programación estructurada (como C). Presenta la ventaja de que no requiere compilación, sino que cualquier *script* en texto plano puede ser directamente "ejecutado" por el intérprete.

Además, uno de los principales objetivos del diseño de este lenguaje es la sencillez de extensión. Se pueden escribir fácilmente nuevos módulos en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. Además, los programas escritos en este lenguaje son normalmente mucho más cortos que sus equivalentes en C o C++, por varios motivos, que se enumeran a continuación:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola sentencia.
- No es necesario declarar los argumentos ni las variables.
- Una de las características sintácticas más distintivas de Python, es que utiliza como delimitadores los espacios en blanco. Un bloque es delimitado dependiendo cómo se indenten las líneas, a diferencia de muchos otros lenguajes de programación que suelen utilizar llaves para llevar a cabo este cometido.

Otra ventaja de Python a destacar, es la cantidad de alternativas que ofrece para el desarrollo de aplicaciones web. Existen servidores de aplicaciones complejos y ligeros, plantillas para el desarrollo web y la posibilidad de usar *python* embebido en un documento html. Debido a ello, este lenguaje cada vez está tomando mayor protagonismo en la web. Sus comienzos fueron ejecutando *scripts* individuales sobre el servidor **Apache** a través del módulo *mod-python*, hasta llegar a los "frameworks" actuales.

En este punto, es importante resaltar que el módulo *mod-python* ha tenido una especial relevancia para el desarrollo de este proyecto, debido a que gran parte del código de una de las entidades de la plataforma de gestión de identidad está escrito en este lenguaje. Por lo que se explica brevemente a continuación en qué consiste.

Mod_python es un módulo de Apache que introduce el intérprete de Python en el servidor. Éste permite ejecutar directamente el código escrito en Python en

Apache, de modo que elimina la necesidad de llamar a procesos externos o ejecutar un servidor adicional. Además, se trata de una interfaz para un subconjunto del API de Apache, de modo que ofrece la posibilidad de llamar a funciones internas de Apache desde Python.

Para finalizar, el hecho de que importantes compañías como *Google* utilicen Python para desarrollar aplicaciones como *Google Docs* o de que el conocido portal de *YouTube* esté escrito en su totalidad en este lenguaje, permite mostrar la especial relevancia de este lenguaje en el desarrollo de servicios web.

2.5.1.1. Quixote

Quixote es un marco de trabajo para el desarrollo de aplicaciones Web en Python. Se trata de un paquete de código abierto, bajo la misma licencia que Python. Se basa en un diseño simple y flexible, que permite escribir programas de manera rápida y obtener grandes beneficios del amplio número de módulos ofrecidos por Python. Las aplicaciones basadas en este "framework" presentan un rendimiento excelente.

La arquitectura de las aplicaciones basadas en Quixote consiste principalmente, en un conjunto de módulos agrupados en un único directorio. Quixote asigna una URL a un método de un objeto Python, dicho método se llama con el contenido de una petición HTTP y el resultado se devuelve al cliente.

Quixote se puede conectar a un servidor web de varias maneras:

1. Utilizando un servidor HTTP escrito en Python. Esta opción permite una mayor facilidad a la hora de realizar la configuración y es bastante adecuada por ejemplo, para el despliegue de una intranet o de servicios web a pequeña escala.

2. Mediante un un servidor con soporte SCGI. Esta opción permite actualizar el código de la aplicación sin afectar a otras operaciones que se estén realizando en el sitio Web y es la que ofrece un mejor rendimiento.

3. A través de un servidor con soporte CGI. Esta opción no es muy recomendable, debido a que es necesario crear un nuevo proceso en cada petición.

Algunas de las ventajas que ofrece este marco de trabajo son:

- Se encarga de los detalles de la interfaz con el servidor web, como por ejemplo, analizar las variables de un formulario.

- También lleva a cabo el procesamiento de archivos subidos a la aplicación.

2.5.2. CGI

CGI (en inglés, *Common Gateway Interface*) [37] es un estándar que define un tipo especial de URL. Debidamente interpretada el servidor arranca el programa indicado con los parámetros definidos. La primera forma de creación de contenido dinámico en páginas web fue a través del mecanismo CGI. Surgió como respuesta a las necesidades dinámicas, interactivas, de control o de recuperación de resultados de un determinado proceso, que requieren multitud de aplicaciones, dado que los documentos HTML se caracterizan por ser estáticos.

Es un programa que se ejecuta en tiempo real en un servidor web en respuesta a una solicitud realizada por el navegador de un usuario. Por cada petición, el servidor ejecuta el programa CGI pasándole los datos de esta solicitud, este programa escribe el HTML en la salida estándar y el servidor web la envía al cliente, de tal manera, que los CGI´s permiten la generación dinámica de código HTML dentro de una propia página HTML.

CGI se caracteriza por ser una tecnología multiplataforma, compatible con el 99 % de los sistemas operativos, servidores y clientes web. Sin embargo, presenta la desventaja de que es necesario ejecutar los programas en el servidor, de los cuales es necesario tener una copia por cliente, pues al tratarse de una pasarela, no es posible utilizar los recursos compartidos ni tener acceso a un control directo de la comunicación.

El servidor web puede enviar información al CGI de varias maneras: a través de la línea de comandos, de la URL, de la entrada estándar, etc. En este estándar, se definen tres tipos de variables de entorno:

- **Variables de entorno específicas del servidor**: Proporcionan información acerca de las características del servidor, como por ejemplo, `SERVER_NAME` para indicar el nombre del servidor, `SERVER_PORT`, para indicar su puerto, etc.
- **Variables de entorno específicas del cliente**: Ofrecen información acerca de las características del cliente, como por ejemplo, `HTTP_USER_AGENT`.
- **Variables de entorno específicas de la petición**: Proporcionan información sobre la petición recibida, como por ejemplo, `CONTENT_LENGTH`, `CONTENT_TYPE`, `QUERY_STRING`.

Por último, un programa CGI puede estar escrito en cualquier lenguaje que pueda ser interpretado o compilado. Actualmente, existen multitud de aplicaciones CGI desarrolladas en C, C++, Fortran, Perl, Tcl, Visual Basic, AppleScript,etc.

Capítulo 3

Descripción general del sistema

En este capítulo se pretende proporcionar al lector una visión general acerca de la infraestructura del sistema de gestión de identidad desplegado a lo largo del desarrollo de este proyecto. Se presentan detalles acerca de los requisitos funcionales de nuestro sistema, y las herramientas seleccionadas para proporcionar el soporte requerido.

También se realiza una descripción de la arquitectura y de las funcionalidades obtenidas de los populares casos de uso de *Single-Sign-On* y *Single-Logout*, comentados en el capítulo 2. Al finalizar este apartado, el lector tendrá una noción general del funcionamiento del sistema propuesto, que le permitirá poder abordar los capítulos 4 y 5, en los que se especificarán detalles más técnicos, de configuración y a bajo nivel de los distintos elementos que forman parte del sistema de gestión de identidad.

3.1. Requisitos funcionales

El objetivo que se persigue en el presente proyecto es poner en marcha una infraestructura que nos permita realizar pruebas acerca de la funcionalidad de un sistema IdM (*Identity Management*). A continuación se enumeran los principales requisitos que debe cumplir nuestra plataforma:

- El sistema deberá garantizar de forma robusta los aspectos de seguridad e identificación básicos. Estos aspectos incluyen: autenticación, integridad, confidencialidad, no repudio y auditoría. Para ello, deberá disponer de una Autoridad de Certificación configurada con las políticas adecuadas para emitir certificados a las distintas entidades que participen en la plataforma

desplegada.

- El sistema estará basado en las especificaciones de SAML 2.0.
- El sistema deberá realizar una gestión de usuarios eficiente.
- El sistema contará con entidades que desempeñen los principales roles definidos en las especificaciones de SAML (IdP y SP).
- En el sistema se podrán identificar distintos perfiles para los usuarios. Participarán usuarios finales y usuarios encargados de administrar las entidades que formen parte del mismo.

De cara a realizar pruebas acerca de las distintas funcionalidades que nos puede proporcionar un sistema de gestión de identidad, estudiaremos y verificaremos la interoperabilidad entre los distintos proveedores que formen parte nuestra plataforma. Para ello, realizaremos pruebas de los perfiles de SSO y SLO con distintos "bindings" definidos en las especificaciones de SAML.

3.2. Arquitectura

En la figura 3.1 se muestran los distintos elementos que componen la infraestructura del sistema de gestión de identidad propuesto. Se pueden distinguir dos tipos de componentes: entidades y usuarios con diferentes perfiles (administradores y usuarios finales).

- **Proveedor de Servicios (SP)**

  Un proveedor de servicios, como se explicó en la sección 2.3, se trata de una entidad responsable de proporcionar un determinado servicio a aquellos usuarios o entidades que confían en un proveedor de identidad para llevar a cabo las tareas de identificación. Nuestra plataforma cuenta con dos proveedores de servicios, denotados como SP_1 y SP_2. Un ejemplo de SP puede ser un servicio de alquiler de coches por Internet o una agencia de viajes.

- **Proveedor de Identidad (IdP)**

  Es una entidad de confianza, responsable de emitir, mantener y gestionar la información de identidad relativa a los usuarios en una federación. Posee la infraestructura de autenticación e implementa la funcionalidad para llevar a cabo el almacenamiento de credenciales, el borrado y la recuperación de

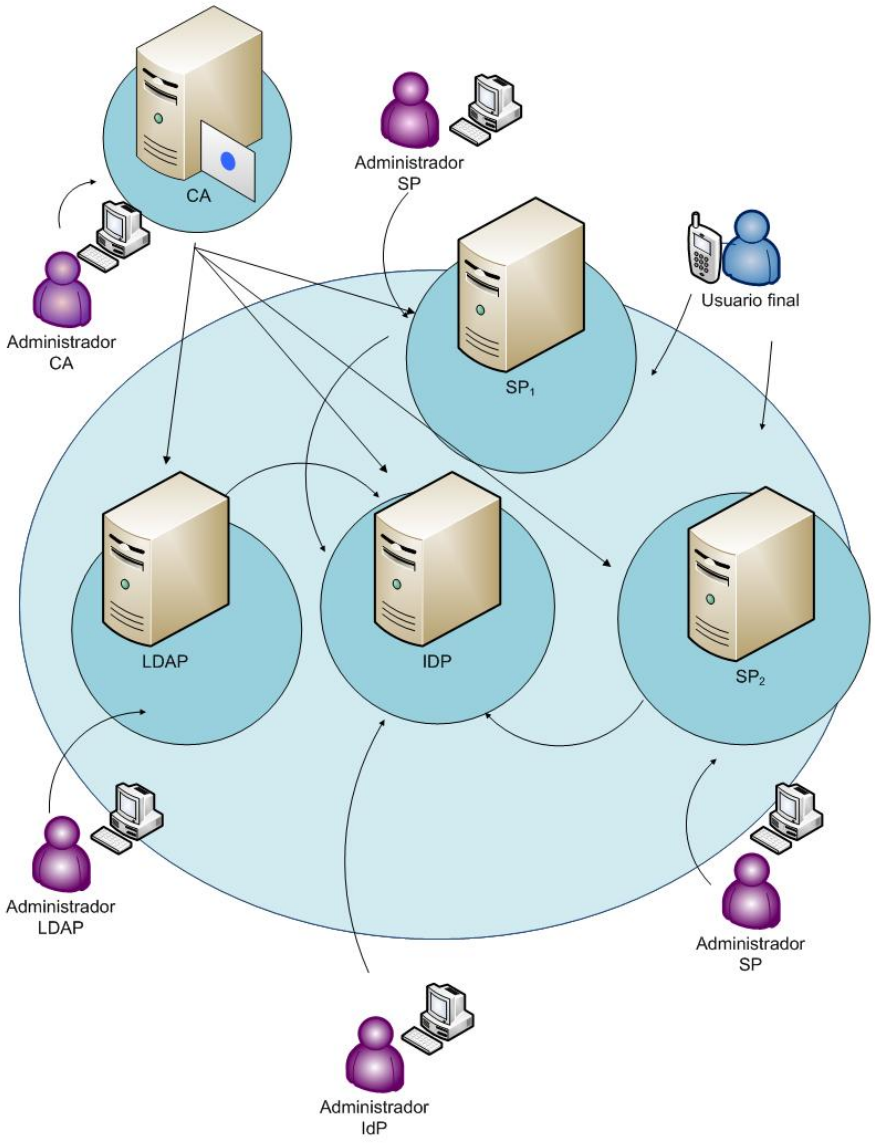

Figura 3.1: Arquitectura general del sistema de gestión de identidad

las mismas por parte del usuario, etc. También se encarga de emitir asertos para que pueda ser utilizado por el SP a la hora de tomar decisiones sobre un Usuario Final. Un ejemplo de proveedor de identidad puede ser una empresa que trabaje conjuntamente con el proveedor de servicios de alquiler

de coches y la agencia de viajes del ejemplo anterior, en la que ambos proveedores confíen; o una universidad que gestiona su propio directorio de cuentas de estudiantes.

- **Autoridad de Certificación(CA)**

  Es la entidad de confianza que da legitimidad a la relación de una clave pública con la identidad de un usuario o servicio. Es la encargada de emitir certificados al proveedor de identidad (IdP) y los proveedores de servicios (SPs).

- **Servidor LDAP**

  El proveedor de identidad dispone de un servicio de directorio LDAP para llevar a cabo las tareas de gestión de los usuarios registrados.

En lo que se refiere a los distintos tipos de usuarios, podemos identificar usuarios que desempeñan tareas de administración de cada una de las entidades comentadas y usuarios finales:

- **Usuario Final**

  Es el que interacciona con los servicios proporcionados por los SPs y también, es el sujeto de los asertos emitidos por el IdP. El Usuario Final puede acceder a los servicios por medio de un agente de usuario como por ejemplo, el navegador de un teléfono móvil para acceder a un servicio web.

- **Usuario Administrador del SP**

  Es el encargado de realizar las tareas de configuración y mantenimiento del SP. Genera sus metadatos para que el IdP y el SP pueden tener acceso a ellos. Configura también los metadatos de otras entidades que quieran establecer una relación de confianza con el SP, de modo que éstas resulten conocidas.

- **Usuario Administrador del IdP**

  Es el encargado de realizar las tareas de configuración y mantenimiento del IdP. Genera sus metadatos para que los SPs pueden tener acceso a ellos. Configura también los metadatos de otras entidades que quieran establecer una relación de confianza con el IdP, de modo que éstas resulten conocidas. También se encarga de tareas relacionadas con la gestión de usuarios, en las que colabora con el Administrador de LDAP.

- **Usuario Administrador de LDAP**

  Es el responsable de la configuración y el mantenimiento del servidor LDAP. Se encarga de dar de alta o de baja a los usuarios del sistema, cuando se lo comunica el Administrador del IdP. También permite al IdP autenticar a los usuarios que acceden a los distintos servicios.

- **Usuario Administrador de la CA**

  Es el encargado de configurar la CA con las políticas adecuadas. Genera certificados digitales para los SPs y el IdP cuando lo solicitan sus respectivos administradores.

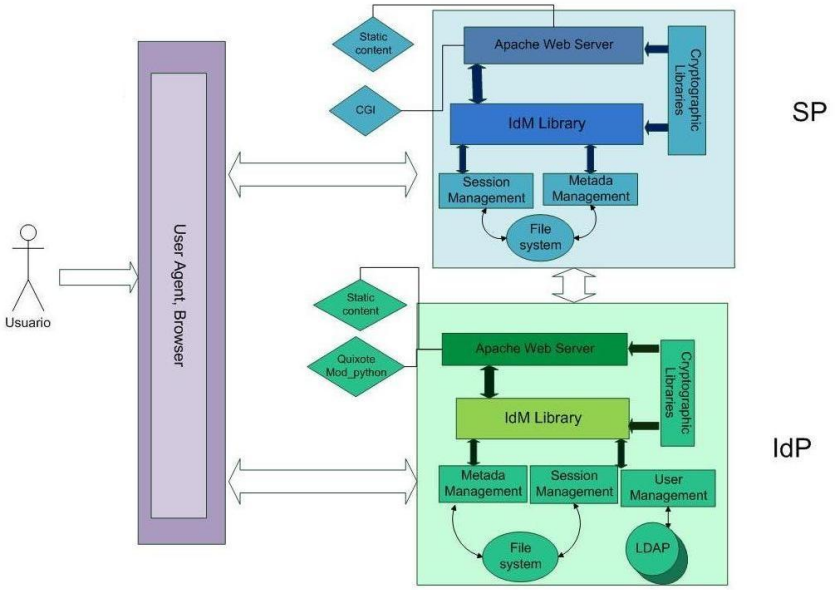

Figura 3.2: Módulos principales del sistema de gestión de identidad

Para finalizar esta sección, en el diagrama de la Figura 3.2 se detallan los distintos bloques que constituyen los principales elementos del sistema de gestión de identidad: el **IdP** y los **SPs**, así como la relación entre los mismos. Aunque cada uno de estos bloques será explicado detalladamente, la siguiente descripción permite obtener una idea básica acerca de su función dentro del conjunto.

- **Módulo *Apache Web Server***: Ambas entidades ofrecen sus servicios a través de un servidor web con soporte SSL. Para que el servidor pueda proporcionar páginas web seguras con el protocolo HTTPS, requiere un

certificado. Este certificado permite que el servidor pueda utilizar cifrado asimétrico para intercambiar claves de cifrado con los clientes, antes de iniciar una transmisión segura de información. Para la generación de este certificado utiliza el módulo ***Cryptographic Libraries***. Además, este servidor nos permite ofrecer páginas web con contenido estático y contenido dinámico a través de CGI, Python y el paquete *Quixote*.

- **Módulo *Cryptographic Libraries***: Es un conjunto de librerías criptográficas que ofrecen varias funcionalidades:

  - La generación de certificados para los servidores web.
  - La comunicación con el módulo ***IdM Library*** para las tareas de cifrado/descifrado, validación/firma de los mensajes intercambiados en el diálogo SAML entre el IdP y los SPs. También contribuye en la generación de metadatos para estas entidades.

- **Módulo *IdM Library***: Ofrece la funcionalidad principal del SP o el IdP. Proporciona el soporte necesario de los cuatro elementos principales definidos en SAML: asertos, protocolos, "bindings" y perfiles. Se comunica con los módulos ***Metadata Management*** , para almacenar los metadatos de los proveedores con los que el IdP o el SP ha establecido una relación de confianza; y con el módulo ***Session Management*** para almacenar identificadores y datos de sesión de usuarios que acceden al IdP o los SPs. Esta información es almacenada en un sistema de ficheros (*File system*). En el caso del IdP, se puede observar que *IdM Library* se apoya además, en el módulo ***User Management***, para autenticar a los usuarios que acceden a sus servicios a través de un directorio LDAP.

Por último, en la Figura 3.2, también se puede observar que la interacción del usuario con el IdP y los SPs se lleva a cabo a través de un agente de usuario (*User Agent*), que en este caso se trata de un navegador web.

3.3. Selección del entorno

En esta sección exponemos las principales características de las distintas herramientas seleccionadas, con el fin de cumplir requisitos identificados en 3.1. Se ha llevado a cabo un estudio de distintas implementaciones de código libre que existen de las especificaciones SAML teniendo en cuenta aspectos de: funcionalidad implementada, versiones de los protocolos soportados, la integración con el

sistema subyacente de gestión de certificados, la complejidad, la disponibilidad para diferentes distribuciones Linux, etc. En las siguientes secciones se exponen los principales aspectos de las librerías e implementaciones estudiadas.

3.3.1. Lasso

Lasso [24](*Liberty Alliance Single Sign-On*) es una librería implementada en C, desarrollada por una compañía francesa llamada Entrouvert, con *"bindings"* para distintos lenguajes como Java, Perl, Python y PHP. Esta librería primero implementó Liberty ID-FF 1.2 pero más tarde añadió soporte para ID-WSF y SAMLv2. El soporte que tiene de los estándares es muy alto.

3.3.2. OpenSAML

OpenSAML [38] ha sido desarrollado por Internet2[6] como una librería para C++ y Java. Está librería implementa las versiones 1.1 y 2.0 de SAML, soportando los asertos, protocolos y *"bindings"*, pero no los perfiles.

OpenSAML es una librería de código abierto que puede ser compilada con un compilador C++, Java o instalada desde paquetes binarios en las plataformas soportadas como Linux, ... Esta librería requiere las siguientes dependencias:

- log4shib or log4cpp
- OpenSSL
- libcurl
- Apache Xerces-C
- Apache XML-Security-C
- XMLTooling-C
- cxxtest (opcional, para soporte de la unidad de pruebas)

Entre las principales ventajas de OpenSAML podemos mencionar la cobertura del estándar, y las bibliotecas que ofrece: `OpenWS` y `XMLToolingLibrary`. *OpenWS* facilita en gran medida el trabajo con Servicios Web a bajo nivel, incluyendo el procesamiento de mensajes SOAP, clientes independientes del transporte para conectar a Servicios Web, y diversos transportes para dichos clientes. *XMLTooling* permite trabajar con fragmentos XML como si fueran Java Beans, al estilo de

JAXB, XMLBeans, o XStream (utilizado en XMPP). La ventaja fundamental de *XMLTooling* es el soporte de firma y cifrado XML.

3.3.3. Zxid

Zxid[22] principalmente implementa un Proveedor de Servicios basado SAMLv2. Es producto es de código abierto y está implementado en C, pero soporta Perl, PHP, y Java mediante SWIG. Además, soporta otros protocolos como ID-FF, ID-WSF y WS-Fed.

3.3.4. Authentic

Authentic [39] es una implementación de código abierto de un Proveedor de Identidad, conforme a las especificaciones de los estándares de Liberty Alliance (ID-FF 1.2 y ID-WSF) y SAMLv2. Utiliza la biblioteca de Lasso y está certificado por el consorcio Liberty Alliance. Además, está recomendado por Zxid para ser interoperable con su SP.

3.3.5. Shibboleth

Shibboleth[7] es una implementación de perfiles construida sobre OpenSAML. Ha sido desarrollado por Internet2, incluye un IdP desarrollado en Java y un SP desarrollado como un módulo C++ para Apache. La versión 1.3 de Shibboleth implementa un IdP y un SP de acuerdo con la especificación SAML V1.1 y la versión 2.0 de acuerdo con SAML V2.0.

La tabla 3.1 muestra una comparativa entre las distintas librerías e implementaciones presentadas, en la cual se evalúan aspectos tales, como su complejidad de uso, soporte de estándares, documentación que ofrecen, etc.

Complejidad	Soporte	Documentación	Otras observaciones
Lasso			
Media-alta. Incluye 420 ficheros escritos en C.	Proporciona una alta cobertura de las especificaciones ID-FF 1.2, ID-WSF y SAMLv2. Soporta asertos, protocolos y *"bindings"* y perfiles.	Está bien documentada. Ofrece un API de referencia de los principales métodos y estructuras, guías de configuración e instalación, además de ejemplos de (W)SPs e IdPs. También proporciona demos para probar los servicios ofertados. Cuenta con una lista de correo para dar soporte a usuarios no de pago. Añade resultados de pruebas de interoperabilidad con otros productos como Zxid, Lightbulb y SourceID.	Está escrita en C, pero permite trabajar también con otros lenguajes, como Java, Perl, Python y PHP. Está recomendada por Zxid para ser interoperable con sus productos.
OpenSAML			
Muy alta. La versión de C++ incluye un total de 802 ficheros y la versión de Java cuenta con **16** bibliotecas , **46** paquetes, **1281** clases, **20** ficheros XML, **40** esquemas. Además, contiene en su parte de validación y pruebas **308** clases y **327** ficheros XML.	Implementa versiones 1.1 y 2.0 de SAML. Soporta asertos, protocolos y *"bindings"*, pero no perfiles.	Está muy bien documentada. Proporciona APIs de sus clases Java/C++. Manuales de usuarios y desarrolladores. También cuenta con manuales de instalación para sus dependencias externas y dispone de una lista de correo para proporcionar soporte a usuarios no de pago	No proporciona soporte perfiles.
Zxid			
Media-Alta. Incluye un total de 297 ficheros C.	Soporta las versiones 1.1 y 2.0 de SAML y cubre parcialmente las especificaciones de Liberty ID-FF 1.2 y las versiones 1.1 y 2.0 de Liberty ID-WSF. Del que mayor cobertura presenta es de SAML 2.0. Proporciona un SP en varios lenguajes (C, Java y PHP). Soporta asertos, protocolos, "bindings" y periles SAML. Roles IdP, Servidor de Descubrimiento y WSP no implementados.	Facilita documentación para instalación y configuración del SP. Ofrece APIs de referencia para los distintos estándares.	Es una implementación en C, pero permite trabajar con otros lenguajes como Perl, PHP, y Java mediante SWIG. Recomienda *Authentic* como IdP interoperable con su SP.

continua en la siguiente página

Complejidad	Soporte	Documentación	Otras observaciones
Authentic			
Media. Incluye un total de 61 ficheros escritos en Python.	Soporta estándares ID-FF 1.2, ID-WSF y SAMLv2. Es un IdP basado en Lasso. Permite posibilidad de emitir asertos de autenticación, autorización y atributos. Soporta gestión de usuarios en sistema de ficheros, LDAP y PostgreSQL y permite habilitar servidor proxy para trabajar con otros IdPs. Dispone de interfaz gráfica de configuración.	Está bien documentado. Ofrece un manual de instalación y configuración. También dispone de una demo. Hay una lista de correo, propia para Authentic, además de la de Lasso, para dar soporte a usuarios no de pago.	Está escrito en Python, pero es compatible con otros lenguajes como C, Java, PHP y Perl. Recomendado por Zxid.
Shibboleth			
Media-alta	Soporta las versiones 1.1 y 2.0 de SAML. Implementación de perfiles basada en OpenSAML. Ofrece IdP en Java y SP en C++.	Está bien documentado. Facilita manuales de instalación del IdP y el SP. Tiene un API público de clases para el IdP. Información del proyecto Shibboleth, especificaciones técnicas, demostraciones, etc. Cuenta también con una lista de correo y manuales para desarrolladores.	Es una implementación orientada a entornos universitarios.

Tabla 3.1: Tabla comparativa de librerías e implementaciones de código abierto de SAML/Liberty

En esta tabla, se puede observar que en general todas las librerías e implementaciones analizadas presentan un nivel de documentación bastante aceptable. Por lo que si nos fijamos en el resto de características, se puede observar que tanto OpenSAML como Shibboleth, soportan únicamente las especificaciones de SAML, aunque hay que resaltar que la cobertura del estándar es muy amplia. La librería Lasso y las implementaciones Zxid y Authentic, ofrecen la posibilidad de trabajar también con las especificación de Liberty.

En lo que se refiere a la complejidad, encontramos que OpenSAML presenta la gran de desventaja de contar con un número considerable de clases y bibliotecas, lo cual hace que resulte muy complicada de utilizar. Además, tampoco ofrece soporte de perfiles SAML. La librería Lasso en cambio, implementa distintos perfiles y soporta también las especificaciones de Liberty en aproximadamente la mitad de archivos que OpenSAML. Por otro lado, las implementaciones de Zxid y Authentic nos ofrecen el soporte del SP y el IdP, respectivamente, en un número de ficheros razonable.

Por tanto, para la realización de este proyecto, se han seleccionado dos marcos de trabajo: **ZXID** y **Lasso**. ZXID proporciona el soporte necesario para el SP

que en la sección 3.2 denotamos como SP_1, y Lasso permite desarrollar tanto SPs como IdPs. Cabe destacar, que el otro proveedor de servicios del sistema, SP_2, lo hemos implementado con ayuda de la librería Lasso. Los motivos de esta elección son los siguientes:

- Son implementaciones de código abierto desarrolladas en C/C++.
- Soportan las especificaciones SAML 2.0 y Liberty ID-FF e ID-WSF 2. El hecho de que estos marcos de trabajo proporcionen soporte también de esta última, nos permitirá realizar pruebas de interoperabilidad con otros proveedores basados en Liberty Alliance en trabajos futuros.
- Ambos marcos de trabajo utilizan como librería criptográfica OpenSSL, lo cual posibilita una fácil integración con otros sistemas como por ejemplo, OpenCA y extensiones de OpenSSL, puesto que SAML no estandariza todos los aspectos en la gestión de identidad como son, por ejemplo, los mecanismos para el establecimiento y gestión de cuentas y privilegios asociados, autenticación, control de acceso, etc.
- Otra de las razones que llevó a elegir estos "frameworks", es que ambos permiten realizar pruebas de interoperabilidad entre las distintas entidades de acuerdo con los roles que soportan.

3.3.6. Servicios adicionales

Para disponer de una solución completa desde el punto de vista tecnológico, los marcos de trabajo seleccionados requieren la instalación y configuración de servicios adicionales, que se describen a continuación:

- Para proporcionar los servicios de autenticación, integridad, confidencialidad y no repudio, el modelo de seguridad del sistema se basará en tecnologías de clave pública/privada para la protección de mensajes a través de su cifrado, y el estándar X.509 para el intercambio seguro de dichas claves públicas. Para ello, emplearemos una librería criptográfica llamada **OpenSSL**, de código abierto, que nos permitirá crear nuestra propia Autoridad de Certificación y generar certificados digitales.
- Los servicios del SP y del IdP se ofrecen a través de un servidor web, por lo cual será necesaria la instalación de un servidor web capaz de soportar conexiones HTTPS. Aunque en la documentación de ZXID se recomienda

utilizar una versión simplificada del servidor Apache y más sencilla denominada *mini-httpd*, se ha seleccionado el servidor **Apache** en su versión más reciente (Apache 2.2), debido a las ventajas que éste aporta entre las cuales, cabe destacar:

1. Soporta una gama amplia de funcionalidades.
2. Se puede reutilizar la instalación para crear distintas instancias de SPs y/o IdPs.
3. Se dispone de gran cantidad de documentación y está soportado por una gran comunidad de usuarios y desarrolladores.

- Por otro lado, para obtener el soporte necesario de otro de los elementos del sistema, el servidor LDAP, se han seleccionado dos marcos de trabajo: **OpenLDAP** [40] y **phpLDAPadmin** [41]. OpenLDAP es una implementación de código libre que proporciona el soporte necesario para desarrollar clientes y servidores LDAP. En lo que se refiere a phpLDAPadmin, es una herramienta para la administración de servidores LDAP escrito en PHP, basado en interfaz Web. Los motivos que llevaron a la elección de estas herramientas son los siguientes:

  - OpenLDAP implementa el protocolo LDAP en su última versión (LDAPv3) y además, soporta una amplia gama de funcionalidades.
  - Dispone también de gran cantidad de documentación y está soportado por una gran comunidad de usuarios y desarrolladores.
  - PhpLDAPadmin ofrece una vista jerárquica basada en árbol, que permite navegar por toda la estructura de directorio. Además, proporciona el soporte necesario para ver los esquemas LDAP, realizar búsquedas, crear, borrar, copiar y editar entradas LDAP.
  - Sólo requiere un servidor Web (en este caso se podrá reutilizar la instalación y configuración de Apache, realizada para proporcionar instancias de SPs e IdPs) y PHP con la extensión de OpenLDAP.
  - Ambos "frameworks" trabajan en varias plataformas de las distintas distribuciones de Linux y Windows, lo cual permite acceder al servidor LDAP desde cualquier lugar en Internet usando un navegador Web.
  - Al tratarse de un interfaz Web, facilita al administrador del directorio LDAP tareas como por ejemplo, la introducción de distintas entradas, la realización de búsquedas, etc. que deberían realizarse mediante la línea de comandos.

3.4. Funcionalidad

El sistema desplegado nos permitirá realizar pruebas con distintos perfiles de SAML. En este apartado se realiza una descripción acerca del soporte que nos proporciona cada entidad SAML de los distintos tipos de asertos, protocolos, "bindings" y perfiles definidos en este estándar, de los cuales el lector puede encontrar una breve descripción en la sección 2.3.1 del capítulo 2.

3.4.1. Funcionalidad del Proveedor de Identidad (IdP)

El proveedor de identidad emite asertos de autenticación y atributos, que permiten a SP_1 y SP_2 tomar decisiones acerca de un usuario final. Los asertos de autenticación permiten informar a los SPs si el usuario ha sido autenticado por el IdP. Con los asertos de atributos, éste puede transmitir información acerca de los atributos de un usuario (por ejemplo que un usuario pertenece a un determinado grupo con más privilegios).

En lo que respecta a los protocolos, esta entidad soporta tres de los principales definidos en SAML:

- **Protocolo de petición de autenticación**: Mediante este protocolo el IdP puede mandar respuestas a mensajes de petición de autenticación enviados por los SPs para autenticar a un usuario. Estas respuestas contienen un aserto de autenticación y pueden contener uno o varios asertos de atributo.
- **Protocolo de registro único de salida (en inglés, *Protocol Single Logout*)**: Mediante este protocolo el IdP podrá realizar un cierre de sesión simultáneo de todas las sesiones asociadas a un determinado usuario.
- **Protocolo *Artifact Resolution***: Este protocolo permite al IdP enviar los asertos por una referencia llamada *Artifact*.

El proveedor de identidad soporta dos tipos de "bindings": `SOAP` y `HTTP-Reditect`. También nos proporciona la funcionalidad de los perfiles de SSO web y SLO. Soporta el caso de uso en el que los procesos de inicio y cierre de sesión únicos son iniciados por el SP. De modo, que nuestro IdP nos permite trabajar con los siguientes perfiles descritos en el resumen técnico de SAML [2]:

- **Perfil SSO Web iniciado por el SP: *Redirect*-POST "binding"**.
- **Perfil SSO Web iniciado por el SP: POST-*Artifact* "binding"**.

- **Perfil SLO, SOAP "binding"**.
- **Perfil SLO, *Redirect* "binding"**.

3.4.2. Funcionalidad de los Proveedores de Servicios (SPs)

Los SPs proporcionan servicios a un usuario final basándose en la información proporcionada por el IdP a través de los asertos de autenticación y atributos. Ambos SPs soportan los siguientes protocolos:

- **Protocolo de petición de autenticación**: Mediante este protocolo los SPs envían mensajes de petición de autenticación al IdP y obtienen sus respuestas.
- **Protocolo de registro único de salida**: Este protocolo permite a los SPs realizar un cierre de sesión simultáneo de todas las sesiones asociadas a un determinado usuario.
- **Protocolo *Artifact Resolution***: Este protocolo permite a los SPs recibir asertos por referencia del IdP y enviar nuevas solicitudes para obtener los valores de dichos asertos.
- **Protocolo de gestión de identificadores**: El proveedor de servicios donatado como SP_1 en 3.2, soporta dos tipos de identificadores para referirse a los usuarios: **"transitorios" (*transient identifiers* o *one-time identifiers*)** y **"persistentes" (*persistent identifiers*)**. Los primeros permiten garantizar que cada vez que un usuario accede a un servicio durante una operación de SSO, el acceso sea anónimo. Estos identificadores son creados para su uso durante una sesión y se destruyen al final de la misma. De esta forma, SP_1 no podrá determinar si un usuario es el mismo que accedió anteriormente a su servicio basándose sólo en el identificador federado. Por otro lado, los identificadores persistentes proporcionan una federación de tipo permanente, es decir, que se mantiene activa hasta que es borrada de manera explícita. En cuanto al proveedor de servicios denatado como SP_2, soporta sólo identificadores de tipo "transitorios".

Los proveedores de servicios de nuestra plataforma soportan distintos tipos de "bindings" definidos en SAML. Ambos ofrecen soporte de los "bindings": `HTTP-Redirect`, `HTTP-POST`, `HTTP-Artifact` y `SOAP`. SP_1 dispone de dos tipos

de “bindings” adicionales: `PAOS` y `HTTP-POST-SimpleSign`. También nos proporcionan la funcionalidad de los perfiles de SSO web y SLO. Y al igual que el IdP, los SPs soportan el caso de uso en el que estos procesos son iniciados por el SP. De modo que, ambas entidades nos permite trabajar con los siguientes perfiles y casos de uso:

- **Perfil SSO Web iniciado por el SP: *Redirect*-POST “binding”**.
- **Perfil SSO Web iniciado por el SP: POST-*Artifact* “binding”**.
- **Perfil SLO, SOAP “binding”**.
- **Federación utilizando identificadores de tipo transitorio (en inglés, *Federation via Transient Pseudonym Identifiers*)**: En este caso de uso se emplea un identificador de carácter temporal para realizar la federación entre el IdP y el SP durante la duración de la sesión de SSO.

El provedor de servicios SP_1 implementa además el **perfil SLO : *Redirect* “binding”** y los siguientes casos de uso:

- **Federación utilizando identificadores de tipo persistente (en inglés, *Federation via Persistent Pseudonym Identifiers*)**: En este caso de uso, la federación de la identidad del usuario se realiza como parte del intercambio de mensajes en el procedimiento de SSO. El proveedor de identidad federa la identidad del usuario en el propio IdP con la identidad del usuario en el SP, utilizando un identificador SAML de tipo persistente.
- **Terminación de la federación**: Este procedimiento permite indicar que un identificador persistente que ha sido previamente establecido en un proceso de federación, dejará de ser utilizado. Es decir, se elimina el acuerdo entre dos proveedores para referirse a un usuario. Sin embargo, cabe resaltar que este caso de uso no se puede probar en nuestro sistema de gestión de identidad debido a que el IdP no lo soporta.

3.4.3. Otras funcionalidades

- Para que los procesos descritos en las secciones anteriores se puedan llevar a cabo, es necesario establecer una relación de confianza entre el IdP y los SPs. Dicha relación se consigue a través de metadatos. Los perfiles definidos en SAML requieren acuerdos entre las entidades del sistema en relación con los identificadores, “bindings” soportados, certificados, claves, etc. Gracias a

las especificaciones de metadatatos de la iniciativa mencionada, es posible describir esta información de forma estandarizada. En dichos metadatos se refleja información acerca de la identidad del proveedor, certificado del mismo, atributos de autoridad, consumidor de asertos, puntos de decisión de políticas, etc.

- El IdP deberá poder acceder a los metatados de los SPs, y del mismo modo, los SP deberán tener acceso a los metadatos del IdP.

- Una vez que la confianza se ha establecido, es necesario un intercambio de material criptográfico: cada entidad (SP e IdP) debe obtener el certificado de la otra parte, puesto que el diálogo SAML entre SP e IdP acerca de un usuario contiene mensajes que llevan firmas digitales y fragmentos cifrados para garantizar seguridad y privacidad.

- Para llevar a cabo la gestión de los certificados que figurarán en los metadatos de estas entidades y los certificados de los servidores a través de los cuales éstas ofrecerán sus servicios; se dispone de una autoridad de certificación configurada con las políticas adecuadas para emitir dichos certificados.

- Por otro lado, la gestión de usuarios se lleva a cabo de dos formas distintas, en función de la entidad. En el caso del IdP, cuenta con un servicio de directorio LDAP para llevar a cabo esta tarea; mientras que los SPs, almacenan los datos de los usuarios registrados y sus sesiones activas en un sistema de ficheros. En este punto, cabe resaltar que se contempla como una mejora futura del proyecto el soporte de LDAP en ambos proveedores de servicios.

3.5. Caso de uso

El sistema de gestión de identidad desplegado, por sus características puede ser bastante apropiado en entornos corporativos, en los cuales, no tiene sentido que los usuario se autentiquen ante cada aplicación si ya se han autenticado inicialmente en el sistema (al iniciar sesión en la red corporativa, por ejemplo). También puede resultar muy necesario en organizaciones en las que debido a su crecimiento, el número de repositorios de identidad, usuarios y servicios haya alcanzado un tamaño lo suficientemente grande como para que sea complicado continuar cumpliendo con las reglas de negocio, enfocadas a la identidad, de la organización sin dedicar excesivas cantidades de recursos a ello.

Además, en Internet se pueden encontrar ejemplos de inicio y cierre de sesión únicos para los servicios de una misma empresa, por ejemplo, Google permite

acceder de forma transparente a sus diferentes servicios (Gmail, Google Calendar, Google Docs,etc.) mediante una única autenticación por parte del usuario.

En lo que se refiere a los casos de uso de la aplicación, podríamos considerar dos supuestos: la realización de SSO y SLO por parte de usuarios sin permisos de administración; y la realización de tareas de administración del sistema del sistema de gestión de identidad por parte de usuario que disponga de los permisos necesarios.

- **CASO 1**:

  1. Un usuario sin privilegios de administrador se conecta desde un navegador web con su teléfono móvil a un proveedor de servicios SP_1. El usuario rellena un formulario presentado en la interfaz gráfica de SP_1, en el cual debe introducir una serie de datos personales (nombre, apellidos, dirección, número de teléfono,DNI).
  2. El usuario pulsa el botón de *Login* y decide realizar la SSO con el "binding" `HTTP-REDIRECT`.
  3. El SP_1 envía una petición de autenticación al IdP.
  4. En caso de que ésta sea correcta, se redirige al usuario a una interfaz de login del proveedor de identidad. Si dicha petición no es válida, el motivo se debe a que el proveedor de servicios no tiene establecida una relación de confianza con el IdP. El administrador del sistema es quien se encargará de resolver el problema.
  5. El usuario introduce un nombre de usuario y una clave.
  6. Si éste está registrado en el servidor LDAP, el IdP es capaz de autenticarlo.
  7. El IdP procesa el mensaje de petición enviado por el SP (atributos del usuario, realiza una verificación de la firma XML) y envía un mensaje de respuesta de autenticación.
  8. El SP procesa el mensaje recibido y en caso de éxito, almacena en un sistema de ficheros los datos de sesión del usuario y se completa la SSO.
  9. El usuario es redirigido a una página en la cual se le indica que la SSO se ha llevado a cabo correctamente y puede realizar un SLO.
  10. Si a continuación, el usuario desea conectarse a los servicios de otro proveedor, SP_2, cuando pulse el *Login*, éste le dará acceso a sus servicios, sin ser redirigido previamente al IdP para introducir sus credenciales. Para ello, se lleva a cabo un intercambio de mensajes entre SP_1 e IdP similar al descrito en los pasos 3, 7, 8 y 9.

En la Figura 3.3 se muestra un diagrama sencillo que permite ver la interacción entre los distintos elementos que intervienen en este caso de uso.

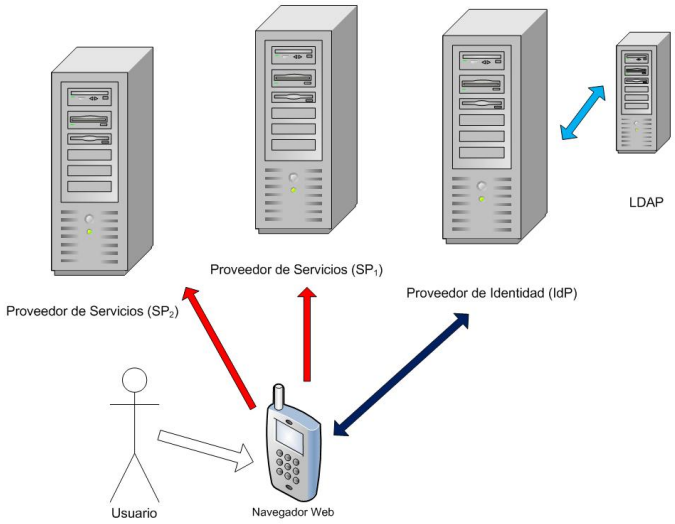

Figura 3.3: Diagrama del Caso de Uso 1

- **CASO 2**:

  Una de las persona responsable de llevar a cabo las tareas de administración del sistema de gestión de identidad, el "Administrador del IdP" desea conectarse al proveedor de identidad, desde el PC de su mesa trabajo.

  1. Realiza la configuración de los metadatos del proveedor de identidad. Para ello, debe proporcionar a la aplicación los ficheros que contienen la clave privada y el certificado del IdP.
  2. Realiza una configuración que permita establecer relaciones de confianza con otras entidades. Para ello, proporciona a la aplicación las URLs a través de cuales el IdP puede acceder a los metadatos de estas entidades, que podrían ser los SPs u otro proveedor de servicios en el que se haya decidido confiar.
  3. También debe contactar con el "Administrador de LDAP" para informarle de que debe dar de alta en el servidor LDAP a un determinado número de usuarios, que podrían ser por ejemplo, los empleados de una organización corporativa u otro empleado al que se desee proporcionar permisos de administración del sistema.

En la Figura 3.4 se muestra un diagrama sencillo que representa este caso de uso.

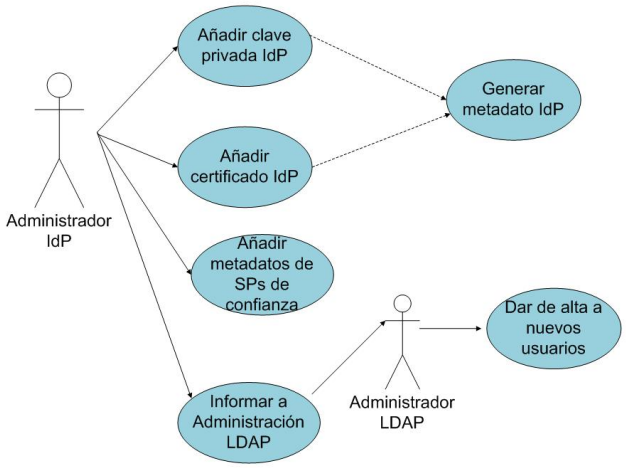

Figura 3.4: Diagrama del Caso de Uso 2

Capítulo 4

Puesta en marcha de la infraestructura de gestión de identidad

4.1. Introducción

En este capítulo se detalla el proceso necesario para llevar a cabo la configuración e integración de dos elementos principales en el escenario de gestión de identidad: el **proveedor de servicios** y el **proveedor de identidad**. Como puede verse en el capítulo 3, se realizó un estudio de las principales implementaciones de código libre existentes de SAML. Los marcos de trabajo seleccionados para proporcionar el soporte necesario para el SP y el IdP, son ZXID y Lasso por los motivos expuestos en la sección 3.3.

Los servicios del SP y del IdP se ofrecen a través de un servidor web, por lo que es necesario disponer de un servidor capaz de soportar conexiones HTTPS. Para ello, hemos realizado una instalación del servidor Apache, en su versión más reciente (Apache 2.2), con dos *Host* Virtuales: un *host* virtual para el SP y otro para el IdP. También, hemos proporcionado al servidor soporte de SSL, de tal forma, que las conexiones establecidas puedan realizarse de modo seguro.

Además del servidor Apache, los "frameworks" seleccionados, requieren la instalación de dependencias externas adicionales, que iremos comentando a largo de este capítulo. En el apéndice B, el lector puede encontrar una descripción acerca del procedimiento a seguir para instalar el servidor con soporte SSL, así como el resto de dependencias externas de ZXID y Lasso.

En las siguientes secciones, nos centraremos por tanto, en explicar los pasos

necesarios para poner en marcha el SP y el IdP, y conseguir el establecimiento de una relación de confianza entre ambas entidades, de modo que puedan interactuar. Finalmente, dedicamos la última sección de este capítulo para describir el funcionamiento de los distintos perfiles que podemos probar con esta infraestructura.

4.2. Despliegue del Proveedor de Servicios (SP)

ZXID ofrece implementaciones de SPs escritas en C, Java, Perl y PHP. En este proyecto se ha utilizado la versión escrita en C, debido a que se trata de un lenguaje modular y fácilmente portable, de modo, que es posible adaptar los programas escritos en C para otros sistemas. Para poner en funcionamiento un Proveedor de Servicios con el soporte de ZXID versión 0.29, se requiere instalar las siguientes dependencias externas:

- Apache con soporte SSL (versiones 1.3 o 2.x)
- Zlib: es una biblioteca de compresión de datos.
- Openssl (versión 0.9.8 o superior).
- Libcurl (versión 7.15.X.X o superior): es una librería que permite la transferencia de archivos y soporta diversos protocolos, entre ellos SOAP el cual es necesario para el soporte de servicios web.

El proceso de configuración del servidor web requiere una cierta complejidad, por lo que dedicamos la siguiente sección para este propósito.

4.2.1. Configuración de Apache para el SP

4.2.1.1. Configuración de Apache con SSL

Durante el proceso de configuración del servidor web seguro para el SP, es indispensable disponer de certificados digitales. Para ello, hemos utilizado un ejemplo de demoCA de OpenSSL que crea una nueva Autoridad de Certificación (CA), la cual permite firmar cuantos certificados deseemos. En el apéndice C el lector puede encontrar un manual detallado acerca de cómo crear una CA y generar certificados digitales utilizando OpenSSL.

A continuación, hemos creado un directorio para almacenar el certificado y la clave privada, que sólo debe ser accesible por el administrador del sistema. El certificado emitido y la clave privada generada para el SP son almacenados en una carpeta que contiene los ficheros de configuración del mismo, llamado *$HOME/Apache/SP*, de tal manera que este directorio tiene la jerarquía representada en la Figura 4.1:

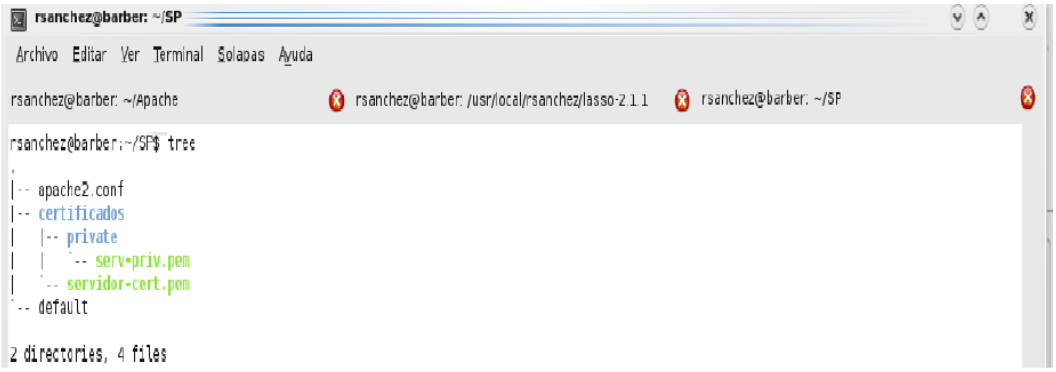

Figura 4.1: Esquema del directorio del SP

En la jerarquía se puede observar que existen dos ficheros de configuración del servidor, uno para los *hosts* virtuales y otro para las directivas generales, que se llaman respectivamente *default.conf* y *apache2.conf*. A continuación se muestra el contenido del primer archivo de configuración mencionado:

default.conf

```
NameVirtualHost *:8443
<VirtualHost *:8443>
  ServerAdmin rsanchez@inv.it.uc3m.es
  ServerName sp.gast.it.uc3m.es
  DocumentRoot /home/apt/rsanchez/Apache/SP/htdocs
  SSLEngine on
  SSLCertificateFile /home/apt/rsanchez/Apache/SP/certificados/
                                        sp-cert.pem
 SSLCertificateKeyFile /home/apt/rsanchez/Apache/SP/certificados/
                       private/sp-key.pem
  <Directory "/home/apt/rsanchez/Apache/SP/htdocs">
     Options Indexes FollowSymLinks MultiViews
     AllowOverride None
     Order allow,deny
     allow from all
  </Directory>
</VirtualHost>
```

En este fichero podemos apreciar una serie de directivas, de las cuales, cabe destacar los siguientes aspectos:

- `ServerName`: se debe poner el nombre del servidor. Dicho nombre debe ser resoluble por DNS, *hosts*, etc. En este proyecto se ha realizado una configuración monolítica y se ha utilizado la misma máquina para ejecutar ambos servicios (el IdP y el SP), para lo que nos aseguramos de que el administrador de la red da de alta ambos alias para la IP de nuestra máquina, y además nos aseguramos de ello en local. A partir de ahora asumimos que el nombre de nuestro servidor es `sp.gast.it.uc3m.es`.

- `DocumentRoot`: Especifica la carpeta raíz que se ubica en el servidor, desde la que se servirán los documentos. En nuestro caso, este directorio se encuentra en `/$HOME/Apache/SP/htdocs`. Este valor también lo debemos especificar en la sección `<Directory>` en la que se establecen los parámetros de configuración de este directorio.

- Las directivas `SSLCertificateFile` y `SSLCertificateKeyFile` indican la ruta donde se encuentran el certificado y la clave privada del SP, respectivamente.

En lo que respecta al fichero de configuración *apache2.conf*, contempla directivas para configurar el servidor con la librería ZXID y una de sus dependencias externas (LibCurl). Por lo que dedicamos el siguiente apartado para describir el contenido de este fichero.

4.2.1.2. Configuración de Apache con ZXID y LibCurl

ZXID implementa un nuevo módulo de autenticación para Apache, denominado `mod_auth_saml.so`. Tras instalar esta librería, este módulo queda situado en el directorio `/usr/lib/apache2/modules`, con el resto de los módulos proporcionados por Apache, y permitirá al servidor realizar las tareas de autenticación a través del perfil de *Single Sign On* de SAML 2.0. En este proceso, el SP obtendrá la información de autenticación del IdP. Para que esto sea posible, es necesario realizar una serie de cambios en el fichero de configuración de Apache. A continuación, se muestra el contenido del fichero *apache2.conf*, junto con una explicación de las directivas empleadas:

apache2.conf

```
# Configuracion de los scripts cgi que se utilizan de interfaz
para el usuario
AddHandler cgi-script .cgi
 <Location "/zxid">
    SetHandler cgi-script
 </Location>

 <IfModule alias_module>
     # client.  The same rules about trailing "/" apply to ScriptAlias
     # directives as to Alias.
     ScriptAlias /cgi-bin/ "/home/apt/rsanchez/zxid-0.29"
 </IfModule>
# Cargamos la ruta a la libreria Libcurl
LoadFile /usr/lib/libcurl.so.3

# Cargamos el modulo de autenticacion con SAML
LoadModule auth_saml_module
               ../../../../../usr/lib/apache2/modules/mod_auth_saml.so
<Location /pers>
   Require valid-user
     AuthType "saml"
   ZXIDConf "URL=https://sp.gast.it.uc3m.es:8443/protected/saml"
   ZXIDConf "DEFAULTQS=l0https://s-idp.liberty-
                                        iop.org:8881/idp.xml=1%26fp=1"
 </Location>

    <Location /intra>
   Require valid-user
   AuthType "saml"
   ZXIDConf "URL=https://sp.gast.it.uc3m.es:8443/protected/saml"
   ZXIDConf "DEFAULTQS=l0https://s-idp.liberty-
                                        iop.org:8881/idp.xml=1%26fc=1"
   ZXIDConf "REQUIRED_AUTHNCTX=urn:oasis:names:tc:SAML:2.0:ac:
                       classes:PasswordProtectedTransport$urn:
                       oasis:names:tc:SAML:2.0:ac:classes:Strong"
    </Location>

    <Location /protected>
    Require valid-user
    AuthType "saml"
    ZXIDConf "URL=https://sp.gast.it.uc3m.es:8443/protected/saml"
    ZXIDConf "ANON_OK=/pers/"
    ZXIDConf "REDIR_TO_CONTENT=1"
 </Location>
# Include the virtual host configurations:
Include /home/apt/rsanchez/Apache/SP/default
```

Con la directivas `AddHandler` y `SetHandler`, indicamos a Apache que todos

los archivos ejecutables para la interfaz de configuración del SP que empiecen por "zxid" los interprete como ficheros CGI. Por otro lado, para que se cargue correctamente el nuevo módulo de autenticación basado en SAML, debemos indicar dónde se encuentra instalada la librería LibCurl, mediante la siguiente línea:

```
LoadFile /usr/lib/libcurl.so.3
```

Después, cargamos dicho módulo con la directiva `LoadModule` y configuramos una serie de localizaciones, que permitirán al SP interactuar con el IdP para realizar la SSO:

`/pers` : La *Single Sign On* (SSO) de SAML es opcional. El Proveedor de Servicios (SP) en este caso utilizará un Proveedor de Identidad (IdP) por defecto.

`/intra` : Requiere SSO de SAML. El SP en este caso utilizará un IdP por defecto.

`/protected` : Requiere SSO de SAML. El SP en este caso puede utilizar un IdP conocido o tiene la posibilidad de elegirlo. Esta es la localización que se ha utilizado para realizar las pruebas.

Si se introduce en el navegador la siguiente URL: `https://sp.gast.it.uc3m.es:8443/protected` se puede visualizar la interfaz gráfica del Proveedor de Servicios, mostrada en la Figura 4.2. Desde esta interfaz, podemos realizar el proceso de SSO iniciado por el SP. Para ello, ofrece los siguientes botones:

Figura 4.2: Proveedor de servicios federado con ZXID.

- **Login(*any*)**: En la documentación de ZXID no se facilita información acerca del funcionamiento de este botón y las pruebas realizadas a través del mismo no pudieron concluir con éxito, como puede verse en el capítulo 6.

- **Login(A2)**: Permite llevar a cabo el perfil **SSO Web iniciado por el SP: POST-*Artifact Bindings***. El SP inicia la SSO utilizando el "binding" `HTTP-Redirect`, que es el que se utiliza típicamente en este caso.

- **Login(P2)**: Permite llevar a cabo el perfil **SSO Web iniciado por el SP: *Redirect*-POST**.

- **Login(S2)**: Implementa el *binding* de tipo `HTTP-POST Simple Sign`.

Para que estos procesos se puedan llevar a cabo, esta interfaz tiene una sección titulada *Technical Options* que ofrece una lista desplegable para seleccionar el tipo de identificadores utilizados para referirse al usuario: "persistentes" o "transitorios". Además, como se puede apreciar, existe la posibilidad de utilizar un IdP ya conocido u otro IdP que se escoja. Por otro lado, si hacemos "clic" en la URL precedida por la etiqueta `Entity ID of this SP` podemos visualizar el metadato del SP, el cual contiene información acerca de su configuración y material criptográfico, que le permitirá establecer relaciones de confianza con otras entidades, como por ejemplo, el IdP.

Figura 4.3: Proveedor de servicios tras completar SSO con ZXID.

Tras completar el proceso de SSO satisfactoriamente, el SP nos permitirá realizar un cierre de sesión único a través de la interfaz mostrada en la Figura 4.3. Para ello, ofrece cinco botones:

- **Local Logout**: Implementa un cierre de sesión local.

- **Single Logout (Redir)**: Permite llevar a cabo el proceso de SLO iniciado por el SP, utilizando el "binding" `HTTP-Redirect`.
- **Single Logout (SOAP)**: Permite llevar a cabo el proceso de SLO iniciado por el SP, utilizando el "binding" `SOAP`.
- **Defederate (Redir)**: Permite llevar a cabo el caso de uso de terminación de federación, utilizando el "binding" `HTTP-Redirect`.
- **Defederate(SOAP)**: Permite llevar a cabo el caso de uso de terminación de federación, utilizando el "binding" `SOAP`.

Para que los procesos de SSO y SLO se puedan llevar a cabo, es indispensable contar con un proveedor de identidad. Por lo que dedicamos la siguiente sección para describir cómo poner marcha el IdP y posteriormente, en la sección 4.6 realizamos una explicación detallada acerca del funcionamiento de los perfiles SSO Web: *Redirect*-POST "binding" y POST-*Artifact* "binding" y del perfil de SLO.

4.3. Despliegue del Proveedor de Identidad (IdP)

Authentic es un proveedor de identidad recomendado por los desarrolladores de ZXID. Para la realización de este proyecto, se ha utilizado la versión 0.7.1 de Authentic. En esta sección, se describen los pasos necesarios para poner en funcionamiento la aplicación del Proveedor de Identidad basada en la librería Lasso.

En primer lugar, se deben instalar las dependencias externas del módulo Authentic antes de proceder a la compilación e instalación del mismo. Dichas dependencias son las siguientes:

- Apache con soporte SSL (versiones 1.3 or 2.x, se recomienda Apache 2)
- Lasso (versión 0.6.3 o mayor)
- Quixote(versión 2.0 o mayor)
- Mod_python o SCGI (versión de SCGI 1.7 o mayor). Se recomienda utilizar el módulo SCGI.

En el apéndice B se puede encontrar una descripción del procedimiento a seguir para llevar a cabo la instalación de estas dependencias, así como del paquete Authentic.

4.3.1. Configuración de Apache para el IdP con SSL y SCGI

[1]

El servidor Apache con SSL ya se tiene instalado y configurado, porque también la necesitaba el SP. Por lo que en este caso, definimos un nuevo *Host* Virtual para el IdP. Las principales modificaciones se deben realizar en el fichero *default.conf* de Apache, que se muestra a continuación:

default.conf

```
NameVirtualHost *:5443
<VirtualHost *:5443>
      ServerAdmin  rsanchez@inv.it.uc3m.es
      ServerName idp.gast.it.uc3m.es
      DocumentRoot /usr/share/authentic/web
      SSLEngine on
      SSLCertificateFile /etc/apache2/ssl/IdP_cert.pem
      SSLCertificateKeyFile /etc/apache2/ssl/IdP_priv.pem
   <LocationMatch "^/($|admin|liberty|login|logout|theme|\
                        change_password|register|forgot_password)">
    SCGIServer 127.0.0.1:3002
    SCGIHandler On
  </LocationMatch>
  SCGIMount / 127.0.0.1:3002
  <LocationMatch "^/(css|images|js)/.*"
 SCGIHandler off
  </LocationMatch>
</VirtualHost>
```

La configuración de Apache del IdP es muy similar a la del SP comentada en la sección 4.2.1 por lo que se destacarán únicamente aquellos aspectos en los que difiere. Para crear los certificados del IdP, se ha seguido un proceso similar al realizado en la generación de los certificados del SP. Se ha creado una copia del certificado y la clave privada en un directorio llamado *certificados*, situado en */$HOME/Apache/IdP*, de tal manera, que este directorio presenta la jerarquía mostrada en la Figura 4.4.

[1]SCGI (*Simple Common Gateway Interface*): Es un protocolo que fue creado como alternativa a CGI y se trata de un estándar para aplicaciones que utilizan interfaces con servidores HTTP

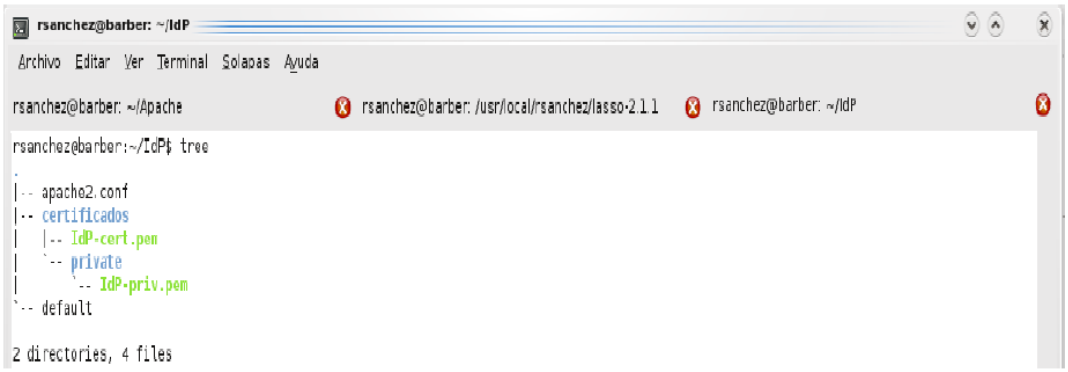

Figura 4.4: Esquema del directorio del IdP

Respecto a las directivas del fichero *default.conf*, cabe destacar los siguientes aspectos:

- `ServerName`: A partir de ahora asumimos que el nombre del servidor del IdP es `idp.gast.it.uc3m.es`
- `Documentoot`: Se encuentra en `/usr/share/authentic/web`, pues en este directorio queda ubicado el fichero *index.html* tras finalizar la instalación de Authentic.
- Además, se ha realizado la configuración del servidor de Authentic con SCGI y se ha indicado el puerto de escucha de éste, que es el 3002. Desde este puerto el IdP se comunica con los SPs para enviar o recibir mensajes de petición o respuesta intercambiados durante un diálogo SAML. Se ha realizado la configuración del servidor con SCGI porque es la se recomienda en la documentación.

4.3.2. Configuración del Proveedor de Identidad

Para iniciar la configuración de Authentic, el primer paso es arrancar los servidores necesarios. Estos servidores son el servidor Apache y el servidor SCGI de Authentic, desde el cual el IdP escuchará las distintas peticiones que lleguen del SP y enviará sus respuestas. Para lanzar el servidor Apache del IdP, utilizamos la configuración definida en el directorio `/$HOME/Apache/IdP`.

Tras este proceso, si se introduce en un navegador web la siguiente dirección: `https://idp.gast.it.uc3m.es:5443/admin`, se puede observar la interfaz gráfica de configuración del IdP mostrada en la Figura 4.5. Para configurar nuestro

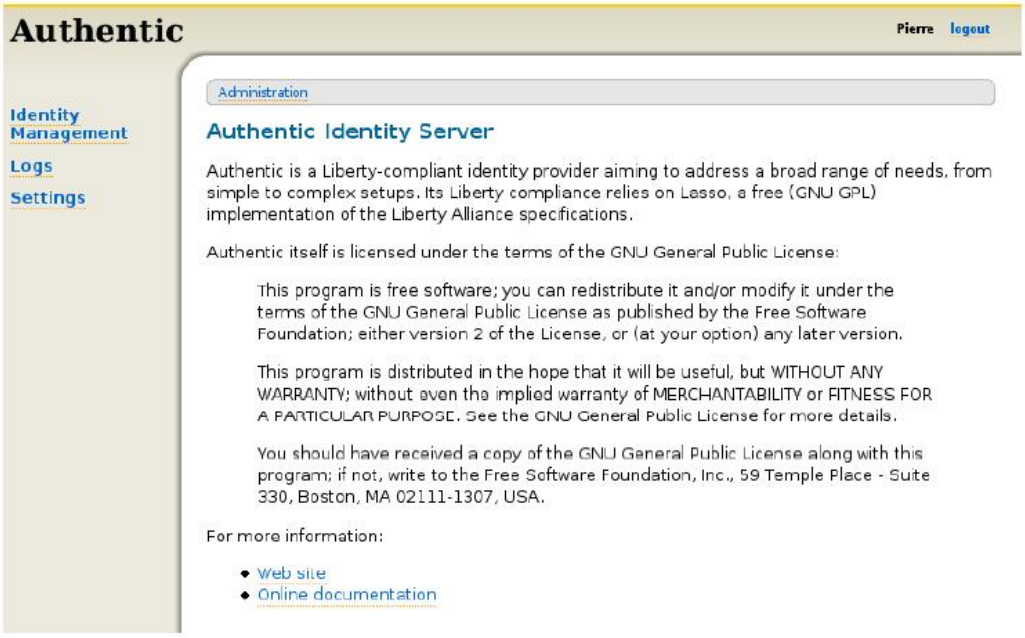

Figura 4.5: Proveedor de identidad con Authentic.

IdP, debemos seguir el enlace: `Settings`→`Identity Provider` y obtenemos la interfaz gráfica de la Figura 4.6.

Los dos primeros campos se rellenan automáticamente y son las direcciones que contienen los metadatos del IdP. Se crean dos metadatos, el primero de ellos sigue las especificaciones de Liberty Alliance y el segundo las de SAML. Como el SP implementado por ZXID sigue las especificaciones de SAML, tendremos que indicarle a éste el segundo de los metadatos para que ambos proveedores sean compatibles.

Los siguientes campos que aparecen son: la clave privada y la clave pública del proveedor, el identificador del mismo (un nombre de usuario, que es una URL), el nombre de la organización que gestiona el Proveedor de Identidad, el dominio común (este campo es útil cuando se asocian varios proveedores de identidad a un proveedor de servicios: se crea una *cookie* en la máquina del cliente y ésta se puede asociar con la identidad del proveedor que la emitió).

Este IdP también tiene soporte de *proxy*. Esta opción permite a un Proveedor de Identidad actuar como un servidor *proxy* entre un proveedor de servicios y el proveedor de identidad final y resulta útil cuando se utilizan varios proveedores de identidad. En nuestro caso, dado que el escenario de gestión de identidad cuenta

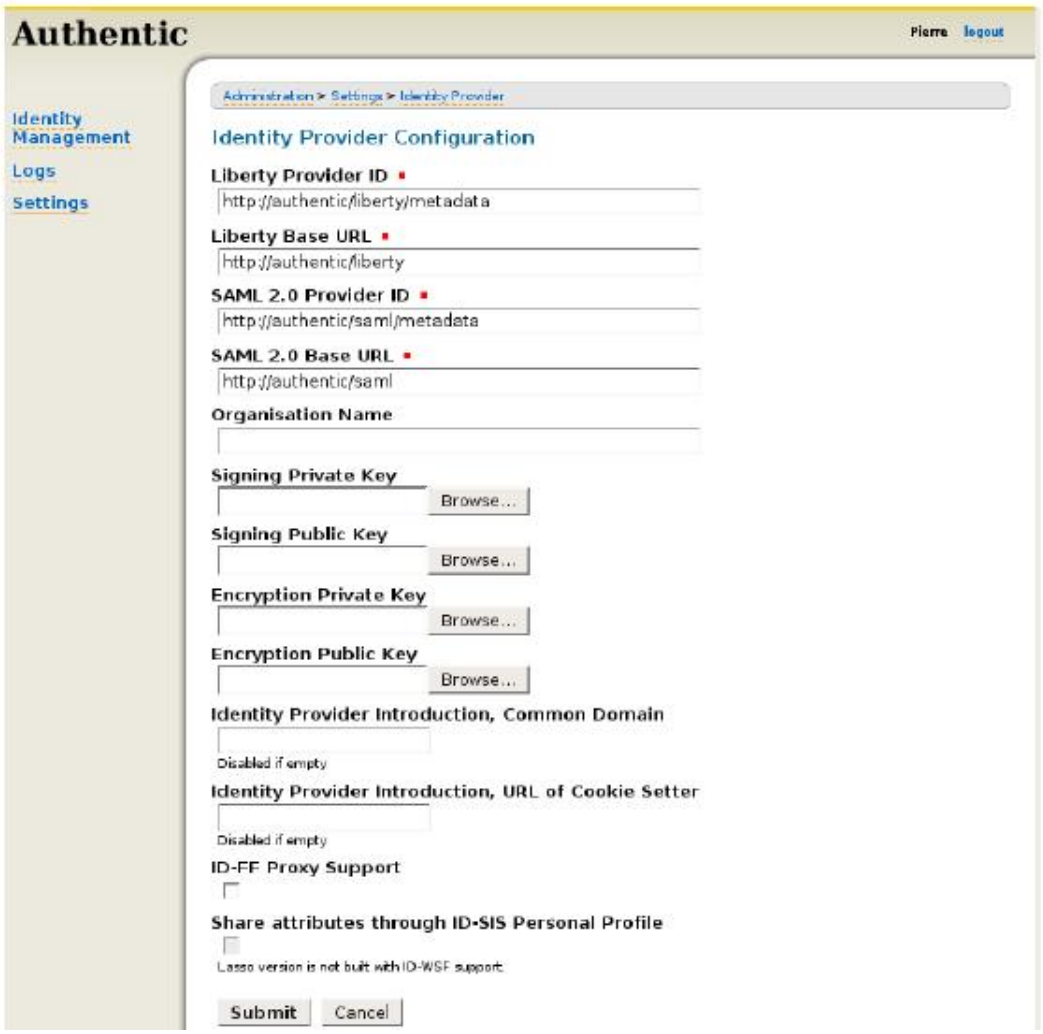

Figura 4.6: Configuración del IdP Authentic.

con un único IdP, no se ha utilizado esta opción, pero se contempla como trabajo futuro introducir nuevas entidades en nuestra plataforma, que se desempeñen también el rol del proveedor de identidad, para lo cual sí necesitaremos habilitar el soporte de *proxy*.

4.4. Establecimiento de una relación de confianza entre el SP y el IdP

Una vez que se tienen configurados los proveedores que van a intervenir en nuestro escenario de gestión de identidad, el siguiente paso, es establecer una relación de confianza entre los mismos. Dicha relación se consigue gracias a los metadatos, los cuales permitirán llevar a cabo un intercambio de material criptográfico. El motivo de que cada entidad (SP e IdP) deba obtener el certificado de la otra parte, se debe a que el diálogo SAML entre SP e IdP acerca de un usuario contiene mensajes que llevan firmas digitales y fragmentos cifrados para garantizar confidencialidad, integridad y autenticidad.

4.4.1. Integración del SP con el IdP

Para realizar esta integración existen dos maneras, la primera es pasar directamente el certificado e información relacionada con el SP y la segunda es introducir simplemente la URL que contiene el metadato. Realizamos pruebas utilizando las distintas opciones con resultados satisfactorios en ambos casos. Sin embargo, cabe resaltar que para poder utilizar la primera opción, el usuario encargado de configurar el IdP, debe tener acceso a los ficheros que contienen el certificado del SP, el certificado de la CA que lo emitió y el metadato del SP. En cambio, si se utiliza la segunda opción, únicamente es necesario conocer la URL que contiene el metadato del SP. Si utilizamos la segunda opción, por ejemplo, es necesario seleccionar:

`Settings→LibertyProviders→Create new from remote URL`

y especificar la URL que contiene el metadato del SP, que en este caso es: `https: //sp.gast.it.uc3m.es:8443/zxid?o=B`

Tras esta operación, el IdP almacena un fichero que contiene el metadato de nuestro SP, en un directorio que se trata del módulo *Metadata Management*, que como puede verse en el capítulo 3, es utilizado por el IdP para almacenar los metadatos de los SPs con los que ha establecido una relación de confianza. Cuando el SP decida establecer una comunicación con el IdP, este último consultará el

directorio en el que almacena los metadatos de los SPs conocidos, para saber si nuestro SP es una entidad en la que confía.

Sin embargo, los primeros intentos de integración resultaron fallidos por el soporte de "bindings" *drafts* dentro de ZXID y porque el IdP no llevaba a cabo la correcta comprobación de uno de sus objetos, que representa un proveedor con el que va establecer una relación de confianza. Para solucionar estos problemas, se han llevado a cabo algunas modificaciones en el código fuente de *Authentic*, como puede verse en el capítulo 6, donde se proporciona un explicación más detallada.

4.4.2. Integración del IdP con el SP

En este caso se sigue un procedimiento similar al punto anterior y para ello, se indica la URL del metadato del IdP en la interfaz del SP de zxid:

```
https://idp.gast.it.uc3m.es:5443/saml/metadata
```

Tras esta operación, el SP almacena un fichero que contiene el metadato de nuestro IdP en un directorio que se trata de su "caché de metadatos" (módulo *Metadata Management*). Cuando el IdP y el SP establezcan una comunicación, el SP realizará una consulta en la caché para comprobar si el IdP es una entidad en la que confía.

Sin embargo, de nuevo, se producen problemas debido a la incompatibilidad en los metadatos. El motivo se debe, a que el metadato del IdP contiene únicamente la clave pública y no el certificado completo, tal como se define en las especificaciones de SAML.

La solución a este problema ha consistido en realizar una serie de cambios en el código del módulo Authentic, con el objetivo modificar la interfaz de configuración del IdP que se mostró en la Figura 4.6 para introducir el certificado en lugar de la clave pública. Así, la parte de creación del metadato de SAML se ha modificado para que cargue un certificado en lugar de una clave pública. Una vez realizada esta modificación, el resultado de integración entre el SP y el IdP ha sido exitoso.

En el capítulo 6 se volverán a tratar los problemas de interoperabilidad que han surgido a lo largo del desarrollo del proyecto y el lector podrá encontrar un mayor nivel de detalle en la explicación de las modificaciones realizadas.

4.5. Gestión de usuarios. Configuración de LDAP

En un sistema de gestión de identidad es fundamental contar con un soporte que permita llevar a cabo las tareas de registro de nuevos usuarios, la asignación de roles y privilegios a los mismos, autenticación, así como el almacenamiento de las sesiones que tienen activas en el sistema, datos de identidad, etc.

Gran parte de estas tareas las lleva a cabo el proveedor de identidad. El SP, por su parte, se encarga de guardar los datos de las sesiones que tienen activas los usuarios registrados en el sistema en cada momento.

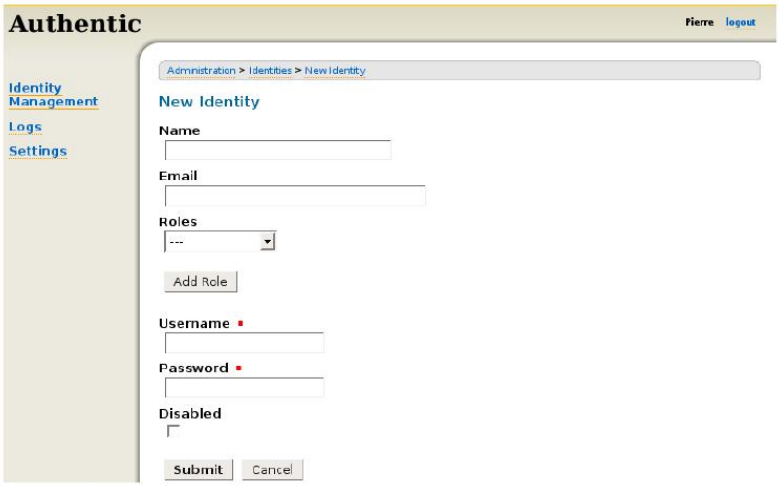

Figura 4.7: Registro de un nuevo usuario en el IdP.

En una primera fase del proyecto, esta información es almacenada en su totalidad en un sistema de ficheros, dado que es la configuración por defecto de Authentic. En la Figura 4.7 se ilustra una interfaz de configuración del IdP que permite crear nuevos usuarios, especificando su nombre, dirección de correo electrónico, nombre de usuario y contraseña. Además, permite también asignar diferentes roles a cada usuario (administrador, usuario normal sin privilegios, etc).

Para ello, seguimos el enlace:

`Identity Management`→`Identities`→`New Identity`

Sin embargo, la idea de almacenar esta información en un sistema de ficheros no resulta eficiente si el número de usuarios que participan en el sistema es elevado. En la documentación de Authentic encontramos que además de gestionar los usuarios a través de sistemas de ficheros, permite soporte para bases de datos

PostgreSQL y servicio de directorio LDAP. Por tanto, dado que gran parte de la información contenida en estos ficheros, es leída o modificada por el proveedor de identidad, se ha instalado un servicio de directorio LDAP para gestionar a los usuarios (módulo *User Management*, como puede verse en el capítulo 3), lo cual, permite mejorar el rendimiento y la escalabilidad de la aplicación.

Para proporcionar el servicio de directorio, se ha llevado a cabo una instalación y configuración de las herramientas de código libre OpenLDAP y phpLDAPadmin. Además, para integrar el código de Authentic con LDAP, es necesario instalar un nuevo paquete llamado **python-ldap** (el código de Authentic está implementado en Python).

Por otro lado, la interfaz de configuración del IdP con LDAP requiere que el tipo de objetos utilizado cuente con los atributos obligatorios *cn*, *uid* y *mail*. Además, tras una exploración de la parte de código encargada de almacenar información en el directorio LDAP, descubrimos que también es necesario que los tipos de objetos almacenados cuenten con el atributo obligatorio *userPassword*.

Por estos motivos, ha sido necesario definir un esquema LDAP propio y crear un nuevo tipo de objeto llamado **userAuthentic** que tiene como atributos LDAP obligatorios: *cn*, *uid*, *mail* y *userPassword*. Además, esta opción nos permite definir nuevos atributos que puedan ser almacenados en el directorio en trabajos futuros, como por ejemplo, información de las sesiones activas del usuario, de identidad, etc. A continuación, mostramos el fichero utilizado para definir la clase de objeto *userAuthentic*:

userauthentic.schema

```
#
# definición del objectclass userAuthentic
# extiende de top.
# Son forzosos los atributos cn, uid, mail y userPassword
# Puede contener los atributos loginShell,uidNumber, etc.

objectclass ( 3.5.0.1 NAME 'userAuthentic'
    DESC 'Entry for an Authentic User'
    SUP top STRUCTURAL
    SUP top AUXILIARY
    MUST ( cn $ uid $ mail $userPassword )
    MAY ( loginShell $ gecos $ description $ uidNumber $ gidNumber
            $ homeDirectory ) )
```

Como se puede apreciar, para definir los atributos obligatorios de esta nueva clase, se debe utilizar la directiva `MUST` y para los atributos opcionales, la directiva `MAY`. Las especificaciones de LDAP permiten que un elemento puede formar parte de varias clases a la vez. Por lo tanto, esta propiedad resulta de gran utilidad para combinar los atributos de las diferentes clases para moldear las entradas del

directorio según nuestras necesidades.

Otro aspecto que debemos consultar en los ficheros de esquemas definidos en OpenLDAP, es el tipo de *objectClass* que se está usando. En la construcción del árbol es obligatorio que los nodos sean de un *objectClass* de tipo `STRUCTURAL`[2], pero además pueden ser de un *objectClass* de tipo `AUXILIARY`[3].

Por último, para que este nuevo esquema quede añadido en la configuración del servidor LDAP, debemos añadir en el fichero de configuración de LDAP (*slapd.conf*) la directiva `include` seguida de la ruta donde se encuentra ubicado el fichero que contiene la definición de la nueva clase. El lector puede encontrar en el apéndice B más detalles acerca del fichero de configuración *slapd.conf*.

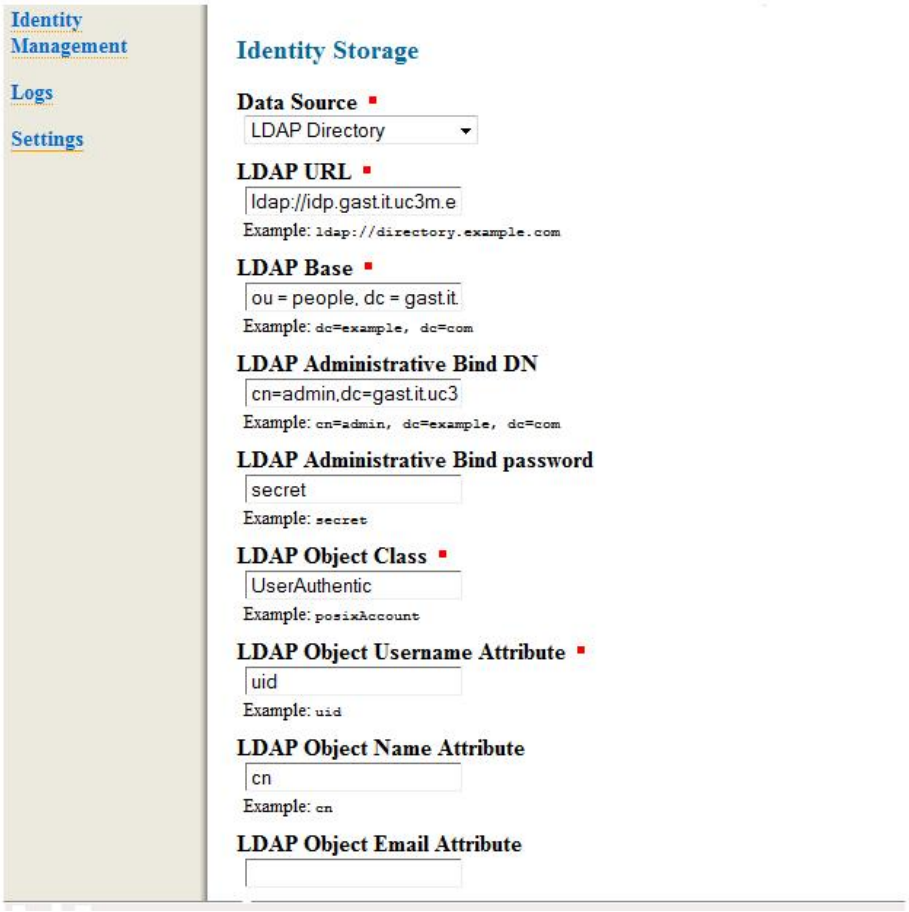

Figura 4.8: Configuración de LDAP en el IdP.

[2]Un *objectClass* del tipo *Structural* define las caracteristicas básicas de un objeto.

[3]Un *objectClass* del tipo *Auxiliary* complementa los atributos de un objeto que ya tiene un *objectClass* de tipo *Structural*. Se utiliza para añadir caracteristicas nuevas de un objeto

Una vez que contamos con este nuevo servicio, es necesario realizar modificaciones en la configuración del proveedor de identidad para que éste soporte la gestión de usuarios con LDAP. Para ello:

`Settings > Identity Storage`

y rellenamos los campos del formulario que se muestra en la Figura 4.8 con los datos de nuestro servidor LDAP. Estos datos son: la URL del servidor LDAP, el nombre distinguido (DN) raíz y administrativo, la clave de administrador, el tipo de clase de objeto que se va a utilizar para almacenar los usuarios en el servidor; que en este caso será **userAuthentic** y los atributos para estos objetos (*cn*, *uid* y *mail*).

Tras realizar esta modificación en el modo de almacenar los usuarios en el IdP, estos se deben dar de alta en el servidor LDAP, debido a que el soporte de LDAP en Authentic no es estable para realizar operaciones de escritura en el directorio. Estas tareas las debe realizar un administrador del sistema, que disponga de los permisos y privilegios adecuados. Las tareas de lectura, en cambio, se realizan adecuadamente. Para ello, si seguimos el enlace: `Identity Management` encontramos una lista de los usuarios registrados en el servidor LDAP. En la Figura 4.9 se muestra un ejemplo con tres usuarios registrados en el sistema. Además, se puede observar que la aplicación también proporciona información acerca de las cuentas de los usuarios.

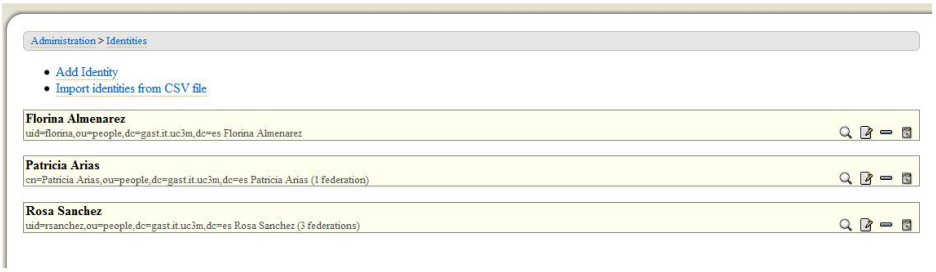

Figura 4.9: Ejemplo de lista de usuarios registrados en el IdP.

4.6. Perfiles soportados

Tras el despliegue realizado, tenemos en funcionamiento dos entidades que soportan los principales roles definidos en SAML: el SP y el IdP. Además, hemos realizado un proceso de integración, de tal modo que se ha establecido una relación de confianza entre ambos proveedores. Por tanto, estamos en disposición de

poder realizar pruebas de los distintos perfiles SSO y SLO, y algunos casos de uso de federación. Estos perfiles ofrecen una gran variedad de opciones en función de la entidad que inicia el flujo de mensajes (SP o IdP) y de los "bindings" utilizados para el envío de mensajes entre SP e IdP. La plataforma de gestión de identidad desplegada en el presente proyecto, permite que los procesos de inicio y cierre de sesión únicos sean iniciados desde el SP y la diversidad de perfiles soportada, radica en la utilización de distintos "bindings" (`HTTP-Redirect`, `HTTP-POST`, `HTTP-Artifact` y `SOAP`) para el intercambio de mensajes. Por tanto, dedicamos esta sección para realizar una descripción acerca del funcionamiento de los perfiles y casos de uso que podemos probar con el SP y el IdP desplegados.

4.6.1. Funcionamiento del perfil SSO *Web*: *Redirect*-POST *Bindings*

En este perfil, el "binding" `HTTP-Redirect` es utilizado por el SP para enviar mensajes de petición SAML (`<AuthnRequest>`) al IdP. Este último, usa el "binding" `HTTP-POST` para mandar el mensaje de respuesta (`<Response>`) que contiene el aserto al SP. En la Figura 4.10 se ilustra el flujo de mensajes intercambiados.

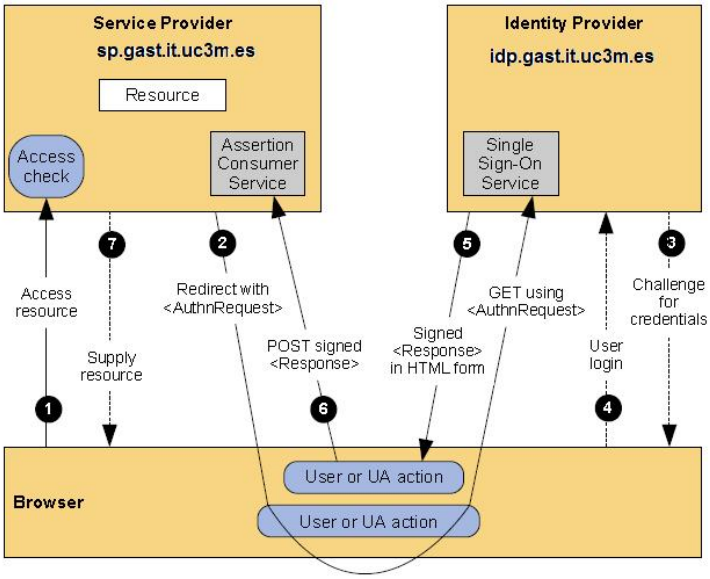

Figura 4.10: SSO iniciada por el SP con "bindings" *Redirect* y POST.

El procedimiento es el siguiente:

1. El usuario intenta acceder a un recurso en `sp.gast.it.uc3m.es`, pero no tiene una sesión válida iniciada, por lo que el SP almacena la dirección del recurso solicitado en información de estado local, que puede ser salvada a través del intercambio de SSO Web.

2. El SP envía al navegador una respuesta HTTP de redirección, que contiene en la cabecera `Location` la URI del *Single Sign-On Service* del IdP (`idp.gast.it.uc3m.es`), junto con una *query string*. Esta *query string* contiene un mensaje de petición de autenticación (`<AuthnRequest>`) codificado. De tal forma, que esta URL presenta el siguiente aspecto: `https://idp.gast.it.uc3m.es:5443/saml/singleSignOn?SAMLRequest=request`

3. El *Single Sign-On Service* determina si el usuario tiene un contexto de seguridad para ese inicio de sesión en el proveedor de identidad. En caso de que no cuente con dicho contexto, el IdP interactúa con el navegador para solicitar credenciales al usuario.

4. El usuario proporciona las credenciales válidas, para ello, el IdP ofrece la interfaz de *Login* mostrada en la Figura 4.11. Después, se crea un contexto de seguridad local de inicio de sesión para el usuario en el IdP.

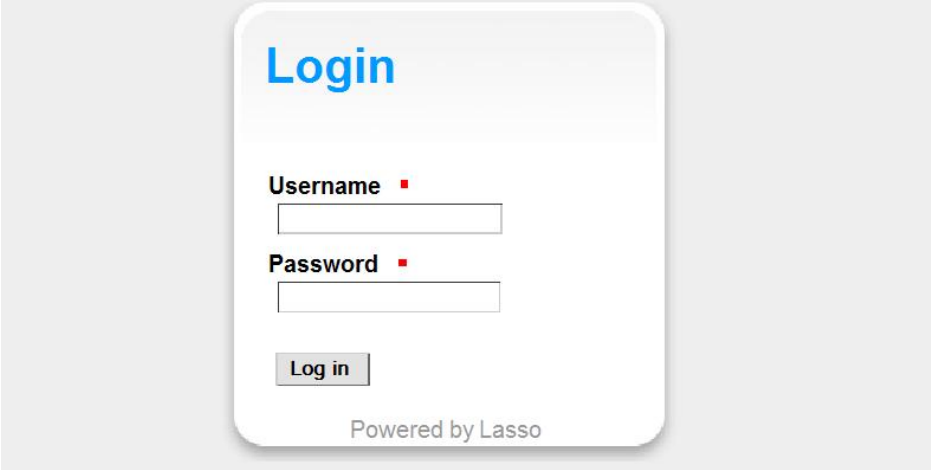

Figura 4.11: Interfaz de *Login* del IdP.

5. El *Single Sign-On Service* del IdP construye un aserto, firmado digitalmente, con la respuesta para el SP. Dicho aserto se envía en un formulario HTML como el valor de un campo llamado `SAMLResponse`. En este caso, se emplea el "binding" `HTTP-POST`.

6. El servicio consumidor de asertos del SP (en inglés, *Assertion Consumer Service*) obtiene el mensaje de respuesta (`<response>`) del formulario HTML para procesarlo. Verifica la firma digital del aserto SAML y luego procesa su contenido para crear un contexto de seguridad local de inicio de sesión para el usuario en el SP. Después, el SP recupera la dirección del recurso solicitado originalmente y envía una respuesta HTTP de redirección a la página de ese recurso.

7. Por último, se realiza un control de acceso para determinar si el usuario tiene autorización para acceder al recurso solicitado.

4.6.2. Funcionamiento del perfil SSO *Web*: POST-*Artifact Bindings*

En este perfil, el SP puede utilizar dos posibles "bindings": `HTTP-Redirect` o `HTTP-POST`, para enviar mensajes de petición SAML al IdP. El SP de ZXID usa el "binding" `HTTP-Redirect`, que además es el que se suele emplear típicamente en este caso. Sin embargo, en algunos casos en los que el tamaño del mensaje es suficientemente grande, es necesario utilizar el "binding" `HTTP-POST`. Por ejemplo, si el mensaje incluye muchos elementos y atributos opcionales, y/o estos deben estar firmados. Por otro lado, el IdP utiliza el "binding" `HTTP-Artifact` para mandar el mensaje de respuesta (`<Response>`) que contiene el aserto al SP. En la Figura 4.12 se ilustra el flujo de mensajes intercambiados.

Como se puede observar, los cuatro primeros pasos son similares a los que se llevan a cabo en el perfil SSO con los "bindings" *Redirect* y POST, y la diferencias se producen en los intercambios numerados como 5, 6 y 7. Por tanto, a continuación realizamos una descripción de los procesos realizados en esta secuencia de mensajes.

En el paso 5, el *Single Sign-On Service* del IdP emite un aserto SAML que representa el contexto de seguridad del usuario para ese inicio de sesión. En este caso, utiliza el "binding" `HTTP-Artifact` para enviar ese mensaje de respuesta (`SAMLart`) a través de una referencia al aserto. Este "binding" permite también, dos mecanismos para enviar el aserto por referencia: a través de redirecciones HTTP o de un formulario HTML. En nuestro caso, el IdP de *Authentic* emplea el primer método para transportar el *artifact*.

A continuación, en los pasos 6 y 7 el servicio consumidor de asertos del SP envía una nueva solitud, a través de SOAP, al servicio de resolución de *Artifact*(*Artifact Resolution Service*) para obtener el valor del aserto. Finalmente, el IdP manda un mensaje de respuesta por SOAP que contiene el valor solicitado.

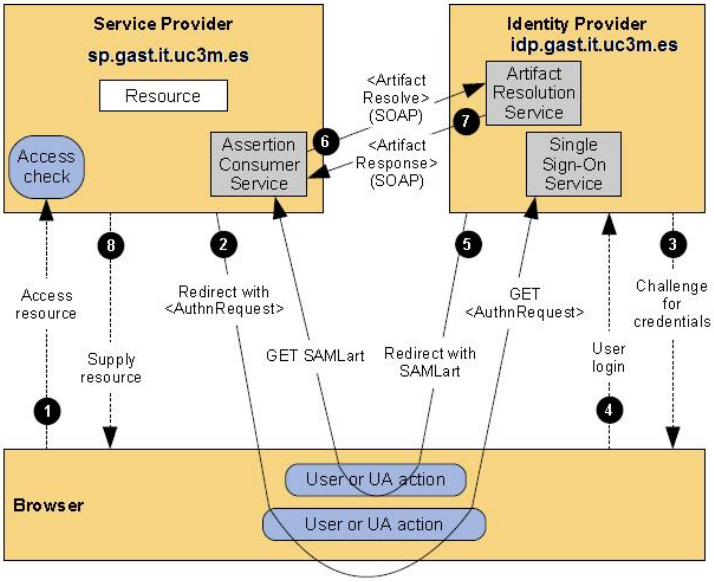

Figura 4.12: SSO iniciada por el SP con "bindings" POST y *Artifact*.

4.6.3. Funcionamiento del perfil *Single Logout*

Una vez que se ha realizado el proceso de SSO, pueden existir varias sesiones individuales en diferentes SPs que comparten un mismo contexto de autenticación. El perfil de SLO permite al usuario terminar una sesión en uno de los proveedores y que automáticamente se cierren todas las sesiones activas en el resto de proveedores.

Este perfil permite además, utilizar ditintos tipos de "bindings": síncronos (`SOAP`) o asíncronos (como por ejemplo, `HTTP-Redirect`, `HTTP-Artifact` y `HTTP-POST`) para realizar el intercambio de mensajes entre el IdP y el SP. En nuestro caso, con los proveedores desplegados, podemos emplear los "bindings" `HTTP-Redirect` y `SOAP`. En la Figura 4.13 se muestra un diagrama en el que representa el diálogo llevado a cabo entre el SP y el IdP durante el proceso de SLO utilizando el "binding" `SOAP`.

En el paso 1, el SP inicia un cierre de sesión único para terminar la sesión del usuario que ha solicitado este servicio. Para ello, manda una petición (`<LogoutRequest>`), firmada digitalmente, al proveedor de identidad del que recibió el aserto de autenticación correspondiente. Esta solicitud puede ser enviada directamente al IdP (a través de `SOAP`) o indirectamente a través de un agente de usuario (por medio del "binding" `HTTP-Redirect`).

En el paso 2, el IdP procesa el contenido del mensaje recibido y verifica si

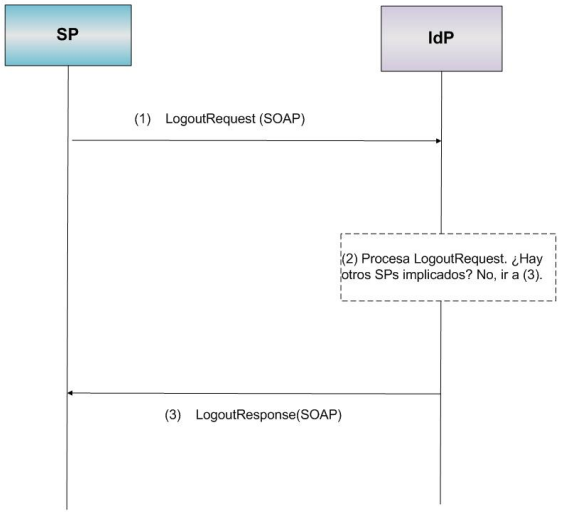

Figura 4.13: SLO iniciado por el SP con el "binding" SOAP.

la petición recibida ha sido enviada por un proveedor en el que confía. Después, determina si se puede finalizar la sesión o si el usuario tiene sesiones activas en otros SPs. Cabe resaltar, que hasta este punto nuestra plataforma dispone de un SP basado en ZXID, por lo que tratamos la primera opción, reflejada en el paso 3, en el que el IdP le emite una respuesta, que puede ser enviada directa o indirectamente a través del navegador. Sin embargo, en el capítulo 5 la posibilidad de SLO con varios SPs resulta de gran interés, dado que hemos implementado un SP adicional. En este caso, el IdP enviaría una solicitud de SLO a este nuevo SP, que le respondería con un mensaje de tipo `<LogoutResponse>`.

Capítulo 5

Implementación de un Proveedor de Servicios

5.1. Introducción

En este capítulo se exponen los detalles de implementación de un nuevo proveedor de servicios, que desarrollamos con el objetivo de explorar la especificación de SAML en más detalle, prestando especial interés en el uso de "binding" SOAP. Además, con este nuevo SP podremos probar el uso de otros "bindings" y realizar pruebas de interoperabilidad con los proveedores desplegados en el capítulo 4. Por otro lado, del mismo modo que el SP de ZXID y el IdP, el proveedor de servicios implementado ofrece sus servicios a través de un servidor web, por lo que se ha reutilizado la instalación de Apache, realizada para la puesta en marcha del SP y el IdP, para crear otra instancia con un nuevo *Host* Virtual.

Para llevar a cabo el desarrollo del proveedor de servicios, se han empleado las librerías de código abierto implementadas en C: Lasso, LibNeon [47] y qDecoder [48]. Dado que teníamos el soporte de Lasso como librería, optamos por utilizarla para implementar las funciones de gestión de identidad basadas en las especificaciones de SAML del SP. La librería *LibNeon*, por su parte, nos proporciona el soporte necesario para implementar el transporte de mensajes SOAP por HTTP/HTTPS intercambiados entre el SP y el IdP. Por otra parte, la librería *qDecoder* nos proporciona funciones que nos resultan de gran utilidad para llevar a cabo tareas de parseo de formularios HTML.

La implementación ha consistido en una librería (***IdM Library***), desarrollada en C/C++ que puede ser utilizada por otros SPs y proporciona el soporte necesario para los perfiles de SSO Web: POST-*Artifact* "Bindings" y SLO (SOAP

"binding") empleando federación con pseudónimos de tipo transitorio, lo que les ofrece la ventaja de "liberarse" de la tarea de mantener y gestionar una base de datos de cuentas de usuarios. Además, hemos implementado una interfaz gráfica de usuario, mediante varios programas CGI escritos en C, que permite que el SP integre fácilmente los servicios ofrecidos por el sistema IdM.

A lo largo de este capítulo se explicará la lógica de las funciones implementadas para la librería IdM y la interfaz gráfica de usuario, así como las estructuras definidas y la relación entre las mismas.

5.2. Librería *IdM*

Esta librería ofrece funciones que permiten implementar el rol del **Proveedor de Servicios** definido en SAML. Las cuatro tareas principales que debe realizar un SP son las siguientes:

1. Crear peticiones de autenticación.
2. Enviar peticiones de autenticación al IdP.
3. Procesar asertos o respuestas de autenticación del IdP.
4. Eventualmente, realizar comprobaciones contra el IdP.

Las dos primeras tareas se llevan a cabo mediante redirecciones HTTP o con el envío de formularios HTML, en función del "'binding" utilizado por el SP para enviar esta petición (`HTTP-Redirect` o `HTTP-POST`). Las dos últimas, se llevan a cabo cuando el IdP realiza una redirección a una URL definida en el fichero de metadato del SP llamada *AssertionConsumerServiceURL*, que se trata del servicio consumidor de asertos. En este caso, la información enviada por el IdP como respuesta a la petición de autenticación, puede estar contenida en la *query string* o en un campo de un formulario HTML, en función del "binding" implementado para llevar a cabo la SSO. Además, como puede verse en el capítulo 4, este mensaje de respuesta contiene el aserto o una referencia al mismo (*artifact*), que permitirá al SP tomar decisiones acerca del usuario. La librería implementada permite recibir los asertos por referencia que llegan desde el IdP, por lo que es necesario realizar una nueva solicitud a través de SOAP para obtener su valor.

El flujo de comunicación llevado a cabo entre el SP y el IdP aparece representado de forma gráfica en la figura 5.1 y permitirá proporcionar al lector una visión general acerca de los mensajes intercambiados entre ambos, y de los cuales se hablará en mayor profundidad a lo largo de las secciones de este capítulo.

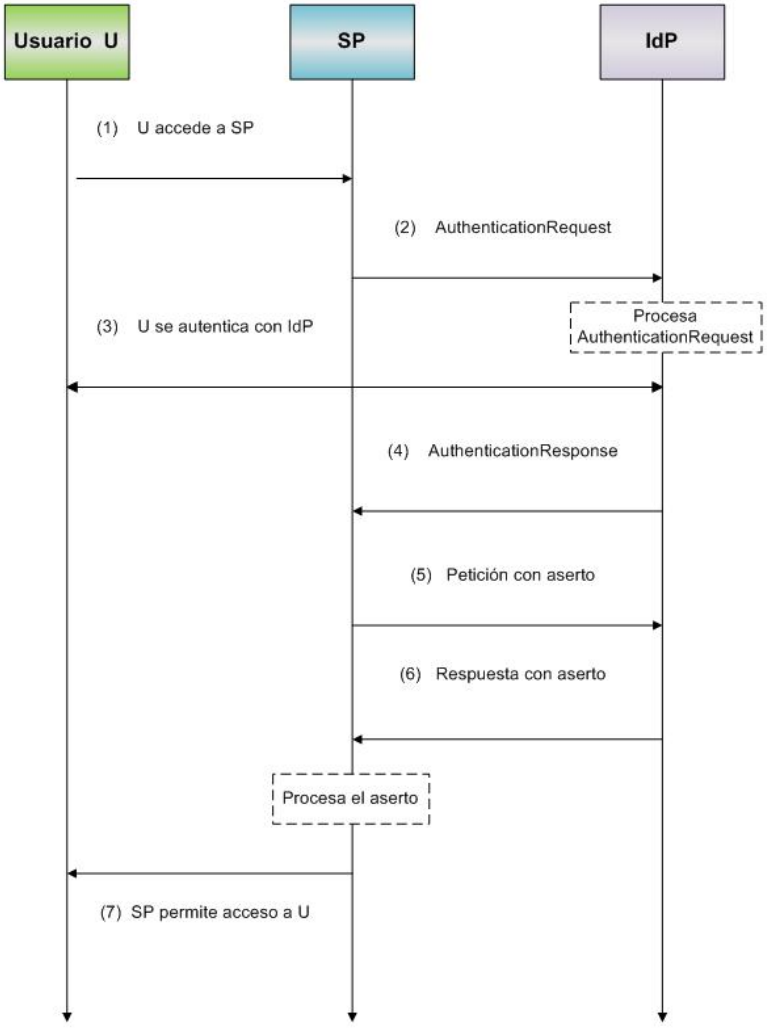

Figura 5.1: Diagrama de mensajes intercambiados entre el SP y el IdP.

5.2.1. Estructura y descripción del API

Para proporcionar una idea global de los métodos definidos en la librería *IdM*, antes de entrar en los detalles de cada módulo de manera independiente, introducimos el diagrama de la Figura 5.2

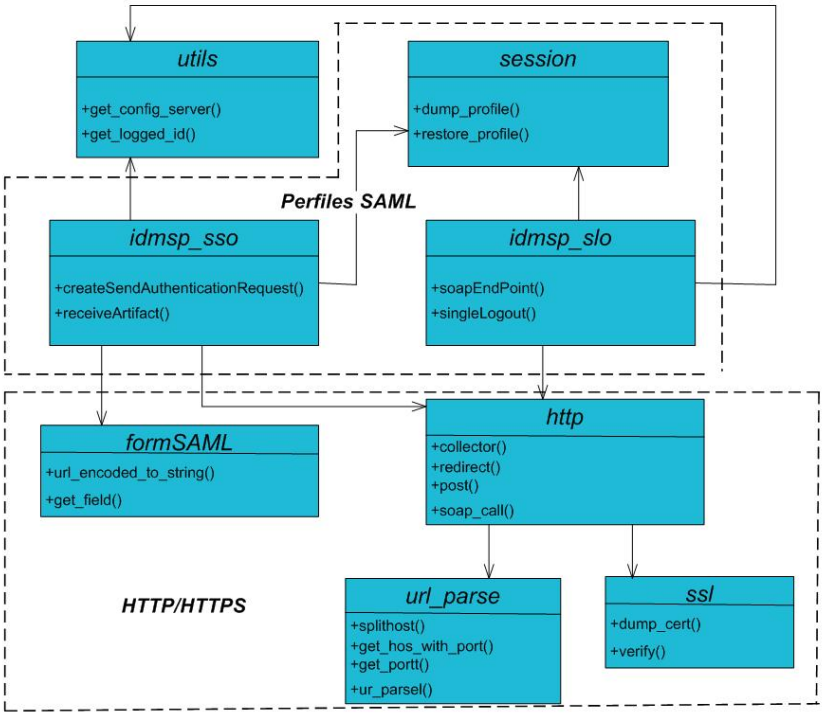

Figura 5.2: Vista de ficheros y métodos de la librería *IdM*.

Los perfiles SAML se han implementado en tres ficheros C denominados *idmsp_sso.c*, *idmsp_slo.c* y *session.c* y además, se han definido sus ficheros de cabecera correspondientes: *idmsp_sso.h*, *idmsp_slo.h* y *session.h*. En cuanto a la funcionalidad de HTTP/HTTPS, se ha implementado en cuatro ficheros C: *http.c*, *url_parse.c*, *ssl.c* y *form_SAML.c*. Para ello, se han definido funciones que nos permiten transportar y procesar los mensajes de petición-respuesta intercambiados entre el SP y el IdP, y obtener información de formularios, en caso de que estos mensajes se envíen o se reciban a través de POST. Por lo que ofrecen soporte para los perfiles SAML.

En cuanto al fichero *utils.c*, dispone las siguientes funciones, que permiten obtener la configuración del SP e información necesaria para poder completar el SLO, como comentaremos más adelante. A continuación, realizamos una breve

descripción de los métodos que constituyen cada fichero.

- En ***idmsso.c*** encontramos las siguientes funciones:

  - ***int createSendAuthenticationRequest (int HTTP_METHOD, char idp_cert)***: Esta función crea un mensaje de petición de autenticación y lo envía al IdP para iniciar el perfil de ***Single-Sign-On* : POST-*Artifact Bindings***. El método recibe como parámetro de entrada:
    - *HTTP_METHOD*: Es el "binding" seleccionado para iniciar el proceso de SSO. Puede ser `HTTP-Redirect` o `HTTP-POST`.
    - *idp_cert*: Representa la ruta donde se encuentra almacenado el certificado del servidor del IdP, de modo que se puedan establecer conexiones seguras a través del protocolo HTTPS.

    La función devuelve cero si la operación se ha completado con éxito o un valor negativo si se ha producido algún error.
  - ***int receiveArtifact(char *idp_cert, char *artifact)***: Esta función recibe un *artifact* enviado por el IdP, lo procesa y vuelve a enviar una solicitud SOAP para obtener el valor del aserto. Si el aserto se procesa correctamente, se completa la SSO y se almacena en un directorio información de sesión del usuario. El método recibe como parámetros de entrada:
    - *artifact*: Representa una referencia al aserto enviado por el IdP.
    - *idp_cert*: Es una cadena de caracteres que contiene la ruta donde se encuentra almacenado el certificado del servidor del IdP, de modo que se puedan establecer conexiones seguras a través del protocolo HTTPS.

    La función devuelve cero si la operación se ha completado con éxito o un valor negativo si se ha producido algún error.

- El archivo ***idmslo.c*** cuenta con los siguientes métodos:

  - ***int soapEndPoint()***: Este método recibe solicitudes SOAP de *Single Logout* enviadas por el IdP, las procesa y envía una respuesta. Devuelve cero si la operación se ha completado con éxito o un valor negativo si se ha producido algún error.
  - ***int singleLogout(char * idp_cert)***: Esta función implementa el proceso de cierre de sesión único utilizando el "binding" SOAP. Recibe como parámetro de entrada la ruta donde se encuentra ubicado

el certificado del servidor del IdP, de modo que se puedan establecer conexiones seguras. El método devuelve como resultado cero si la operación se ha completado con éxito o un valor negativo si se ha producido algún error.

- En el fichero ***session.c*** se han definido las funciones:

  - ***char* dump_profile(LassoProfile *profile)***: Se emplea para almacenar en un fichero un objeto XML que representa el período de sesión de un usuario en caso de que éste haya sido modificado. Recibe como parámetro de entrada una estructura de tipo *LassoProfile*, definida en el API de Lasso [49]. Esta estructura representa un perfil SAML, que en nuestro caso será SSO o SLO y además, cuenta con campos para almacenar el identificador del usuario y datos de sesión El método devuelve un identificador del usuario o NULL si se ha producido algún error.
  - ***void restore_profile(LassoProfile *profile, char *nameid)***: Esta función busca en un sistema de ficheros, con la ayuda de *nameid*, un archivo que contiene información del período de sesión del usuario representado por *nameid*. Hace una lectura del fichero que contiene esta información y la asigna a un campo de la estructura *LassoProfile*.

- En el fichero ***url_parse.h***, ha sido necesario definir funciones que implementen distintas tareas para llevar a cabo el parseo de una determinada URL en sus distintos elementos. Para representar una URL genérica, se ha definido una estructura muy sencilla, que se muestra a continuación:

```
typedef struct urlParsed{
    unsigned int port;
    char *host;
    char *path;
} url_parsed;
```

  Como se puede apreciar, los campos de esta estructura permiten almacenar el *host*, el *path* y el puerto de una determinada URL. Las siguientes funciones permiten dividir una determinada URL en sus distintos componentes:

  - ***char* get_port(char *url, int pos)***: Esta función obtiene el puerto de una determinada URL. Recibe como parámetros de entrada:

- *url*: Es una cadena de caracteres que contiene una URL.
- *pos*: Representa la posición donde se encuentran el carácter ":" a partir del cual se define el puerto de la URL.

El método devuelve una cadena de caracteres que representa el puerto o NULL si en la URL no se especifica el puerto o si se ha producido algún error.

- ***char* get_host_without_port(char *url, int pos)***: Este método obtiene el *host* de una determinada URL. Recibe como parámetros de entrada:
  - *url*: Es una cadena de caracteres que contiene una URL.
  - *pos*: Representa la posición donde se encuentran el carácter ":" a partir del cual se define el puerto de la URL.

  La función devuelve una cadena de caracteres que representa el *host* o NULL si se ha producido algún error.
- ***url_parsed* url_parse(char *url, url_parsed *url_pars)***
  La función parsea una URL en los distintos elementos que la componen (*host*, *path* y puerto) y los almacena en la estructura *url_parsed*. Recibe como parámetros de entrada:
  - *url*: Es una cadena de caracteres que contiene una URL.
  - *url_pars*: Es un puntero de memoria reservada para una estructura de tipo `url_parsed`.

  El método devuelve una estructura de tipo `url_parsed` o NULL si se ha producido algún error durante el parseo de la URL.

- Por otro lado, las siguientes funciones de ***ssl.c*** implementan un *callback*, para llevar a cabo la verificación del certificado del servidor en el caso de que deseemos establecer una conexión segura.

  - ***void dump_cert(const ne_ssl_certificate *cert)***: Esta función recibe como parámetro de entrada una estructura de tipo `ne_ssl_certificate`, que representa un certificado genérico y proporciona información acerca del emisor del certificado.
  - ***static int verify(void *userdata, const ne_ssl_certificate *cert)***: Este método lleva a cabo una verificación del certificado que recibe como parámetro de entrada. Comprueba que éste haya sido emitido por una Autoridad de Certificación de confianza, que el campo *Subject* contenga el *host* del servidor, etc. Recibe como parámetros de entrada:

- *userdata*: Representa el *host* del servidor.
- *cert*: Estructura que representa el contenido de un certificado genérico.

El método devuelve cero si el certificado se ha verificado correctamente y otro valor en caso contrario.

- Las siguientes funciones definidas en ***http.c***, permiten enviar mensajes SOAP a través de HTTP/HTTPS y recibir las respuestas, así como enviar mensajes de redirección o POST:

  - ***char* soap_call(char *url, char *message,char *content_type, char *cert)***: Abre un *socket* para establecer una conexión HTTP o una conexión segura (HTTPS), crea una sesión y envía una petición POST a la URL que se especifique. En el *body* del mensaje de petición con POST se introduce el mensaje SOAP de petición del SP, codificado en base 64. En caso de que la petición se haya realizado con éxito, se procesa el mensaje SOAP de respuesta (enviado por el IdP) y se extrae el *body*. Recibe como parámetros de entrada:
    - *url*: Especifica la URL a la que se debe enviar la solicitud POST.
    - *message*: Representa el mensaje SOAP que se enviará en la petición.
    - *content_type*: Para que la petición se lleve cabo correctamente deberá ser *text/xml* .
    - *cert*: Es el certificado del servidor al que se va a enviar la petición.

    El método devuelve una cadena de caracteres que contiene un mensaje SOAP con la respuesta o NULL si se ha producido algún error.
  - ***int redirect(char *url,char *cert)***: Abre un *socket* para establecer una conexión HTTP o una conexión segura (HTTPS), crea una sesión y envía una petición Redirect. Se utilizará para enviar una solicitud de redirección a la URL que se especifica en el metadato del IdP donde se debe llevar a cabo la SSO seguida de la *query string*, en el caso en el que tratemos de realizar la petición de autenticación mediante HTTP-Redirect. El método recibe como parámetros de entrada:
    - *url*: Representa la URL a la que se debe enviar la solicitud de redirección.
    - *cert*: Es el certificado del servidor al que se va a enviar la petición.

    La función devuelve cero si la operación se a llevado a cabo con éxito o un valor negativo en caso de que se haya producido algún error.

- ***int send_post(char *url, char *message, char *cert)***: Abre un *socket* para establecer una conexión HTTP o una conexión segura (HTTPS), crea una sesión y envía una petición POST. Se utilizará para enviar una solicitud POST a la URL que se especifica en el metadato del IdP donde se debe llevar a cabo la SSO. En el *body* del mensaje de petición con POST se introduce el mensaje de petición de autenticación del SP, codificado en base 64. Esta función es invocada en caso de que se desee realizar la petición de autenticación mediante HTTP-POST. Recibe como parámetros de entrada:
  - *url*: Es la URL a la que se debe enviar la solicitud POST.
  - *message*: Representa el mensaje que contiene la petición de autenticación.
  - *cert*: Es el certificado del servidor al que se va a enviar la petición.

  El método devuelve cero si la operación se a llevado a cabo con éxito o un valor negativo en caso de que se haya producido algún error.
- ***void collector()***: Esta función tiene un carácter auxiliar y su tarea consiste en obtener el *body* del mensaje de respuesta HTTP/HTTPS y almacenarlo en un *buffer*.

- En ***formSAML.c***, encontramos las siguientes funciones, que permiten parsear el formulario enviado por el IdP, en caso de que la respuesta la envíe utilizando el método POST:

  - ***char* get_field(char *field)***: Esta función obtiene el valor del campo *field*, que recibe como parámetro de entrada de un formulario HTML. El tipo de contenido que se debe haber utilizado para codificar los datos del formulario es *x-www-form-urlencoded*. El SP utiliza este método para obtener el valor de la referencia al aserto enviada por el IdP.
  - ***char** urlencoded_to_strings(char *str)***: Es una función auxiliar utilizada por la anterior. El parámetro de entrada *str* representa el formulario obtenido directamente de la entrada estándar. Por lo que se realiza un parseo del mismo y se devuelve un array de punteros en el que cada posición contiene el nombre y el valor de un determinado campo del formulario.

- Por último, en el fichero ***utils.c*** se han definido:

  - ***get_config_server(char *metadata_sp, char *privateKey_sp, char *cert_sp, char *metadata_idp, char *cert_idp, char *ca_chain)***. Recibe los siguientes parámetros de entrada:

- *metadata_sp*: Cadena de caracteres que contiene la ruta en la que se encuentra almacenado el fichero de metadato del SP.
- *privateKey_sp*: Cadena de caracteres que contiene la ruta donde se encuentra almacenada la clave privada del SP.
- *cert_sp*: Cadena de caracteres que la ruta donde se encuentra almacenado el certificado de cifrado del SP.
- *metadata_idp*: Cadena de caracteres que contiene la ruta en la que se encuentra almacenado el fichero de metadato del IdP.
- *cert_idp*: Cadena de caracteres que contiene la ruta en la que se encuentra almacenado el certificado del IdP.
- *ca_chain*: Cadena de caracteres que contiene la ruta en la que se encuentra almacenada la cadena de certificados del IdP.

Esta función devuelve como resultado una estructura de tipo *LassoServer* que almacena el certificado, la clave privada y el metadato del SP. Esta estructura también almacena en una lista la información relativa a los proveedores de identidad conocidos (certificados y metadatos). La función devolverá NULL en caso de que se haya producido algún error al realizar la configuración del SP. Los errores que se pueden producir pueden estar causados porque los certificados, los metadatos o la clave privada del SP sean incorrectos.

- ***char* get_logged_id()***: Esta función obtiene un identificador del usuario de las *cookies* y es utilizada por el método *singleLogout()* como función auxiliar que le ayuda a determinar cuál de los usuarios autenticados en el sistema ha solicitado el cierre de sesión único.

Una vez que tenemos descritas las funciones de la librería *IdM*, en las siguientes secciones explicaremos qué funcionalidades nos proporcionan y cómo se relacionan los métodos entre sí. Posteriormente, realizaremos una descripción del interfaz gráfico de usuario a través del cual, el SP ofrecerá sus servicios.

5.2.2. Inicialización y configuración del Proveedor de Servicios

Para inicializar y configurar el proveedor de servicios, emplearemos la función *get_config_server()*, descrita en la sección anterior. Esta función debe ser capaz de acceder a cierto material criptográfico, que se expone a continuación:

- Fichero que contiene el metadato del SP.

- Fichero que contiene la clave privada del SP.
- Fichero que contiene el certificado x509 *v3* en formato PEM del SP.
- Fichero que contiene el metadato del proveedor de identidad (IdP).
- Fichero que contiene el certificado x509 *v3* en formato PEM del IdP.
- Fichero que contiene la cadena de certificados del IdP.

Para la generación de los certificados y claves requeridos, hemos utilizado un ejemplo de demoCA de OpenSSL, que como puede verse en el apéndice C, crea una Autoridad de Certificación (CA) que nos permite firmar certificados. El fichero que contiene el metadato del IdP lo podemos descargar descargar de la siguiente URL, ofrecida por el proveedor de identidad: `https://idp.gast.it.uc3m.es:5443/saml/metadata.xml` y sigue las especificaciones de SAMLv2.0. En cuanto a la cadena de certificados del IdP, se trata de un fichero que se obtiene como resultado de concatenar el certificado de la CA y el certificado del IdP. A continuación, se detalla el fichero que contiene el metadato del SP:

metadata.xml

```
<EntityDescriptor entityID="https://sp.gast.it.uc3m.es:6443
/metadataSaml.xml">
<SPSSODescriptor AuthnRequestsSigned="true" protocolSupportEnumeration=
                "urn:oasis:names:tc:SAML:2.0:protocol">
<KeyDescriptor use="signing">
  <ds:KeyInfo>
   <ds:X509Data>
     <ds:X509Certificate>
            -----BEGIN CERTIFICATE-----
            MIIDhTCCAm2g ..............
            ............................
            -----END CERTIFICATE-----

        </ds:X509Certificate>
   </ds:X509Data>
  </ds:KeyInfo>
</KeyDescriptor>

<KeyDescriptor use="encryption">
  <ds:KeyInfo>
   <ds:X509Data>
    <ds:X509Certificate>
```

metadata.xml

```
            -----BEGIN CERTIFICATE-----
            MIIDhTCC.....................
            ............................
            -----END CERTIFICATE-----
        </ds:X509Certificate>
   </ds:X509Data>
  </ds:KeyInfo>
</KeyDescriptor>

<SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:
SOAP"
Location="https://sp.gast.it.uc3m.es:6443/wsp?o=0"/>

<AssertionConsumerService isDefault="true" index="0"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
Location="https://sp.gast.it.uc3m.es:6443/wsp?o=A"/>

<AssertionConsumerService index="1"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="https://sp.gast.it.uc3m.es:6443/wsp?o=A"/>

<AssertionConsumerService index="2"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location="https://sp.gast.it.uc3m.es:6443/wsp?o=A"/>
</SPSSODescriptor>

<Organization>
<OrganizationName xml:lang="en"> SP-UC3M</OrganizationName>
</Organization>
</EntityDescriptor>
```

La etiqueta `EntityDescriptor` describe la entidad de nuestro SP y especifica la URL a través de la cual, un proveedor de identidad puede acceder a su metadato. En este fichero también se puede apreciar, que se utiliza el mismo certificado para los propósitos de firma y de cifrado, aunque se pueden emplear certificados distintos para dichos propósitos. Además, cabe resaltar que se reflejan los perfiles y tipos de "bindings" que soporta nuestro SP. Otro aspecto a destacar, es la etiqueta `AssertionConsumerServiceURL` contiene la URL en la cual el SP recibirá los asertos y respuestas de autenticación del IdP.

Los ficheros de metadatos y certificados anteriores se encuentran disponibles en el directorio `$HOME/SP/files` y las funciones de la librería acceden a ellos por medio de una variable llamada `DIR`.

Para finalizar, cabe destacar que para implementar la función *get_config_server()*, nos hemos apoyado en los métodos de Lasso que se

muestran en la tabla 5.1

Función	Descripción
`lasso_server_new ()`	Construye una estructura *LassoServer*
`lasso_server_add_provider()`	Construye una estructura *LassoProvider* y hace que sea conocida por el servidor, añadiéndola a lista de provedores en los que confía

Tabla 5.1: Métodos de Lasso utilizados para la configuración del SP

Tras invocar *get_config_server()* tendremos configurado nuestro proveedor de servicios y se encontrará en su lista de proveedores conocidos el IdP desplegado en el capítulo 4.

5.2.3. Control de acceso

Una vez que tenemos configurado el SP y que éste confía en el proveedor de identidad descrito en el capítulo 4, estamos en disposición de establecer una comunicación entre ambos. En esta comunicación se realizará un intercambio de mensajes entre el SP y el IdP, con el fin de completar el proceso de SSO. Para ello, la librería desarrollada dispone de dos funciones encargadas de enviar y recibir peticiones de autenticación: ***createSendAuthenticationRequest()*** y ***receiveArtifact()***. Para implementar estas funciones, hemos utilizado algunos métodos de Lasso que se muestran en la tabla 5.2.

Función	Descripción
`lasso_login_init_authn_request()`	Inicializa una petición de autenticación a un proveedor de identidad remoto especificado
`lasso_login_build_authn_request_msg()`	Construye una petición de autenticación como un mensaje SAML, o una URL para el *binding* HTTP-Redirect o el valor del campo *SAMLRequest* en un formulario HTML para *binding* HTTP-POST
`lasso_login_build_request_msg()`	Construye un mensaje SOAP que sigue las especificaciones de SAML
`lasso_login_process_response_msg()`	Procesa el aserto que contiene el mensaje respuesta recibido
`lasso_login_process_authn_response_msg()`	Procesa la respuesta de autenticación recibida
`lasso_login_accept_sso()`	Obtiene el aserto de la respuesta y lo añade a la estructura *LassoSession*.

Tabla 5.2: Métodos de Lasso utilizados para la generación, el envío y el procesado de mensajes de autenticación

El procedimiento, ilustrado en la figura 5.3, consiste en generar una petición de autenticación que será enviada al proveedor de identidad. Para construir esta petición, hemos utilizado una estructura definida en el API de la librería Lasso [50], llamada `LassoSamlp2AuthnRequest`, que representa un mensaje de petición

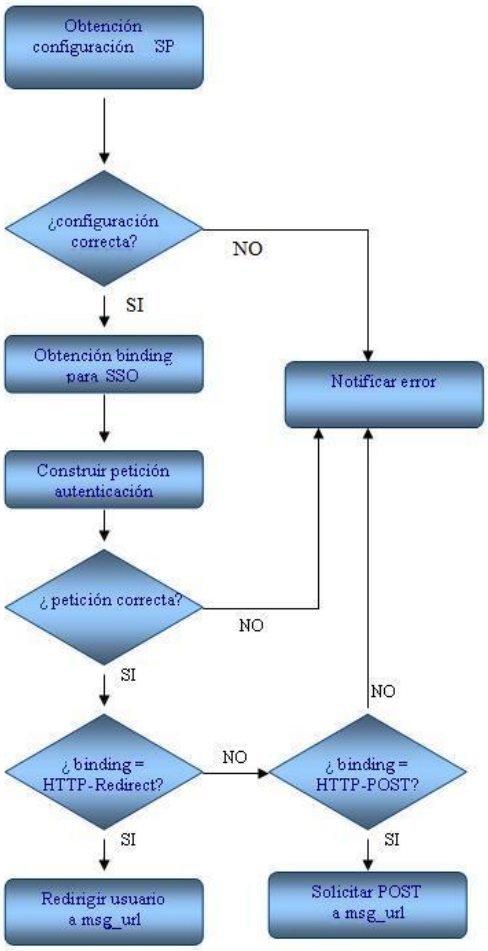

Figura 5.3: Creación y envío de una peticitición de autenticación

de autenticación según las especificaciones de SAMLv2.0. Para construir correctamente la petición, es necesario asignar valores a algunos de los campos de esta estructura, que se detallan a continuación:

- ***nameIDPolicy***: Indica al proveedor de identidad el tipo de política de federación aplicada. Para que podamos utilizar identificadores temporales entre una identidad y el SP, de modo que únicamente sean válidos durante la sesión debemos indicar que este campo tome un valor igual a `LASSO_SAML2_NAME_IDENTIFIER_FORMAT_TRANSIENT`.
- ***IsPassive***: Su valor por defecto es `TRUE` e indica que el IdP no interactuará con el usuario.

- ***ForceAuthn***: Este campo se utiliza únicamente en el caso de que el valor del campo anterior sea `FALSE`. Indica que se forzará la autenticación del usuario, la primera vez que éste acceda los servicios del SP.

En la petición de autenticación generada, el proveedor de identidad no solicitará las credenciales al usuario si ya ha sido autenticado con anterioridad y la federación se creará utilizando identificadores de tipo transitorio.

La función en cargada de generar la petición (*createSendAuthenticationRequest()*), devuelve como resultado cero, en caso de éxito o un valor negativo en caso de que se haya producido algún error en el proceso de generación o envío de la petición al IdP.

Cabecera	**Valor**
GET	/saml/singleSignOn?SAMLRequest=lZFNa8MwDIb%2FSvC9%2BWjSpjVJI DQNFLbRrWOHXYZI3dXg2Jkl7%2BPfz8lgdJfCrrIe%2BXmlAqFXA68dnfW DeHMCKfjslUY%2BPZTMWc0NoESuoRfIqeOH%2BvaGz8OYD9aQ6YxiF8h 1AhCFJWk0C3ZNyV4WeZyt5uvtepsn9WKzSVfNMq%2BzdrVtsrZtMhY8CY u%2Bv2Qe9xCiEzuNBJp8KY7XsySZzdPHJOfJkqf5MwtaYzsxBSoZW SdGau8%2Flu%2BiZCdQKFhVjKp8mmarM9GAPIpwCF8BKZQUui7 tQ%2B%2B%2BzLI06gXBEQgOngl90iK6pIufDd75rLtmb5TsvkaFHuj 6KsaKPM5OUysnCxql0MSCw36cde9AyZMUtmT%2F9WNBrZT52FgB9Bs5qn 60%2F966%2BgY%3D&SigAlg=http%3A%2F%2Fwww.w3.org%2F 2000%2F09%2Fxmldsig%23rsa-sha1&Signature=K1bFpGbo70%2Bv ozkqy4Hllz%2FsK%2BT1cWD%2FWrtDfsDoGbT48RMp%2BvZByORngC m3fq3wplhkYc8UiTWUByv6LmqGCJscinsbJnt2Z5AJYhKm4wW56IHCXaEY s6Xlr%2FXXeNYBZHiPJ%2BHDH2SXX5Anb6BEox530ufObfYzJFM4xjV5Q 5U%3D HTTP/1.1
Accept:	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, */*
Host:	idp.gast.it.uc3m.es:5443
Connection:	Keep-Alive
Accept-Language:	es
Cache-Control:	no-cache
User-Agent:	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
Referer:	https://sp.gast.it.uc3m.es:6443/sp

Tabla 5.3: Solicitud HTTPS de Petición de Autenticación

Esta función es invocada por la interfaz gráfica de usuario, cuando el usuario

pulsa uno de los dos botones de *Login* habilitados (uno que utiliza `HTTP-Redirect` para realizar la SSO y otro que utiliza `HTTP-POST`). En función del botón pulsado, el SP realizará la petición de autenticación con el "binding" adecuado. Si por ejemplo, el usuario solicita iniciar el inicio de sesión único utilizando el "binding" `HTTP-Redirect`, enviaremos una petición como la indicada en la tabla 5.3

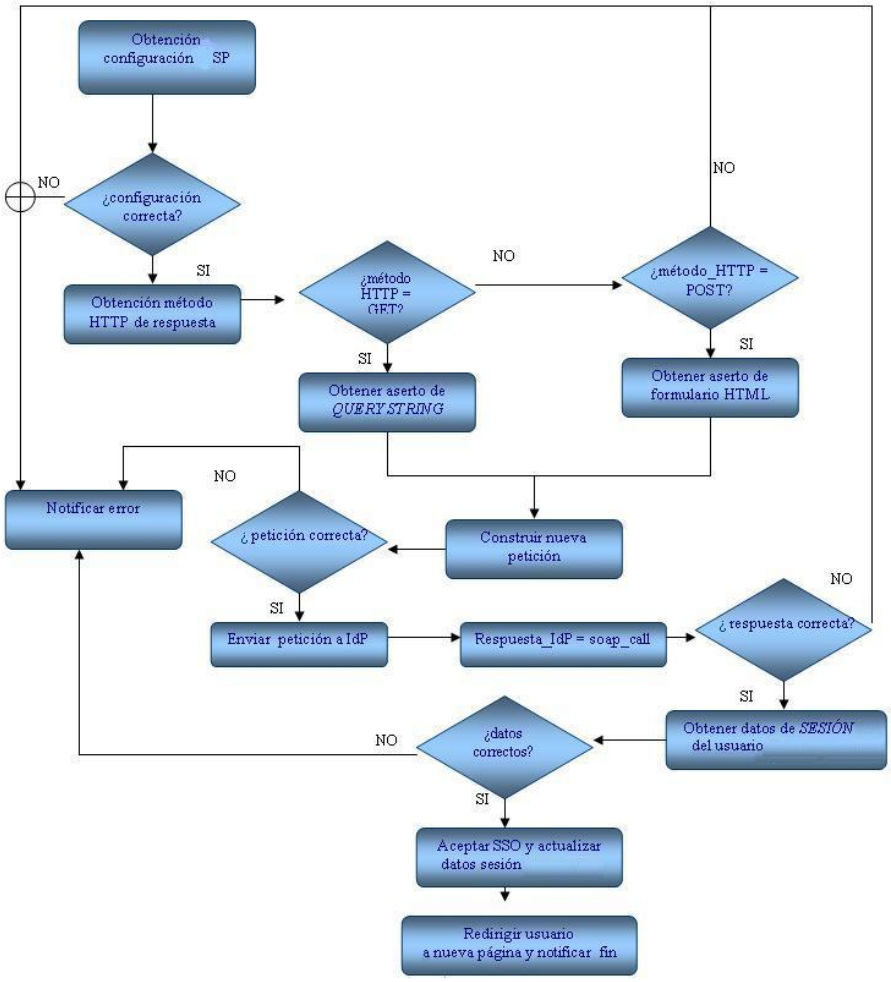

Figura 5.4: Recepción de una respuesta autenticación. Terminación del proceso de SSO

Por otra parte, en la Figura 5.4 se muestra un diagrama de flujo de los procesos llevados a cabo para recibir y procesar los asertos que llegan desde el IdP. El procedimiento seguido es el siguiente: se obtiene la configuración del servidor, y se determina el tipo de método HTTP con el que el IdP ha enviado la respuesta. Si se trata de un GET, el *artifact* coincide con el primer parámetro de entrada y si

se trata de un POST, éste se extrae del valor de un campo denominado *SAMLart* de un formulario HTML. Para obtener este campo, se invoca el *get_field()* de la librería *IdM*.

Una vez que tenemos la referencia al aserto, construimos una nueva petición SOAP, se la enviamos al IdP y obtenemos el valor del aserto. Para obtener esta respuesta, se invoca una de la función *soap_call()*. A continuación mostramos el contenido del mensaje SOAP que se envía al proveedor de identidad, así como la respuesta que recibimos por parte de éste:

Mensaje de solicitud SOAP enviado por el SP

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<samlp:ArtifactResolve xmlns:samlp="urn:oasis:names:tc:SAML:2.0:
protocol" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_E4688B640F0936A52C3E68FEB577F070" Version="2.0"
IssueInstant="2009-11-23T17:53:32Z">
<saml:Issuer>
  https://sp.gast.it.uc3m.es:6443/metadataSaml.xml
</saml:Issuer>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#_E4688B640F0936A52C3E68FEB577F070">
<Transforms><Transform Algorithm=
             "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
  <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>iNhFi9y2SPMXGSM56KCN/ZToSzw=</DigestValue>
</SignedInfo>
<SignatureValue>
c48x+TfF.................
.........................
</SignatureValue>

<KeyInfo>
  <X509Data>
   <X509Certificate>
      MIIDCj...........
        ...................
     </X509Certificate>
 </X509Data>
</KeyInfo>
</Signature><samlp:Artifact>AAQAAAX496IWw+b9xUfHOyRaUFTUF3xSNkQw
 MjExODE1REI3OEM4NjlCRkQ=</samlp:Artifact>
 </samlp:ArtifactResolve></s:Body></s:Envelope>
```

En el mensaje SOAP de petición enviado por el SP, se pueden destacar varios aspectos. Figura el valor de la firma que deberá verificar el IdP entre las etiquetas `<SignatureValue>` y `</SignatureValue>` e información del certificado, definida entre las etiquetas `<X509Certificate>` y `</X509Certificate>`. Además, la etiqueta `<samlp:Artifact>` contiene el valor del *artifact* que envió previamente el IdP. Por lo que enviamos dicho *artifact* al servicio de resolución de *Artifact* del IdP, para obtener el valor del aserto.

Mensaje de respuesta SOAP enviado por el IdP

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<samlp:ArtifactResponse xmlns:samlp=
"urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_38AF5D29F51A54E5B8FD502AF33E98B9"
InResponseTo="_FE250177F95319A2BA6F3F60D5DB7937" Version="2.0"
IssueInstant="2009-11-23T17:26:58Z">
 <saml:Issuer>
  https://idp.gast.it.uc3m.es:5443/saml/metadata
 </saml:Issuer>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
    <CanonicalizationMethod
     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1
 <Reference URI="#_38AF5D29F51A54E5B8FD502AF33E98B9">
  <Transforms><Transform Algorithm=
              "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
  <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>tIVeFehBINrl64NrkkANizqfu4g=</DigestValue>
 </Reference>
</SignedInfo>
<SignatureValue>
wPJ3In ......................
.............................
</SignatureValue>
</Signature>

<samlp:Status>
 <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
 </samlp:Status><samlp:ResponseID="_B506A4BFB392C1BDF5D4D03DDE085556"
   InResponseTo="_B658A7EB1C20BF2699B0680F90CD89E6" Version="2.0"
   IssueInstant="2009-11-23T17:26:57Z">
```

Mensaje de respuesta SOAP enviado por el IdP

```
<saml:Issuer> https://idp.gast.it.uc3m.es:5443/saml/metadata</saml:Issuer>
<samlp:Status>
 <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion Version="2.0" ID="_C310EA9815BB433FBD45E582A79BFB13"
  IssueInstant="2009-11-23 T17:26:57Z">
<saml:Issuer>https://idp.gast.it.uc3m.es:5443/saml/metadata</saml:Issuer>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod
    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#_C310EA9815BB433FBD45E582A79BFB13">
    <Transforms> <Transform Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>DVm4cfBAEyJQoupL+MXoz7H41I4=</DigestValue> </Reference>
 </SignedInfo>
 <SignatureValue>
  kQD8adGm...........
  ...................
 </SignatureValue>
</Signature>
<saml:Subject>
 <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
  NameQualifier="https://idp.gast.it.uc3m.es:5443/saml/metadata">
   2D348A4D2AFC3155B583799BC49D4C05 </saml:NameID>

  <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
   <saml:SubjectConfirmationData Recipient="https://sp.gast.it.uc3m.es:6443/
    sp?o=A"/>
 </saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions><saml:AudienceRestriction>
<saml:Audience>https://sp.gast.it.uc3m.es:6443/metadataSaml.xml
</saml:Audience>
</saml:AudienceRestriction></saml:Conditions>
<saml:AuthnStatement>
 <saml:AuthnContext>
 <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:1.0:am:password
</saml:AuthnContextClassRef>
</saml:AuthnContext></saml:AuthnStatement>
</saml:Assertion>
</samlp:Response></samlp:ArtifactResponse></s:Body></s:Envelope>
```

En lo que se refiere al mensaje SOAP de respuesta del IdP, se puede apreciar que la solicitud del SP se ha procesado satisfactoriamente en el IdP, a través de la etiqueta `<samlp:Status>`, que contiene un código de éxito (*status:Success*). Además, nos devuelve el valor del aserto solicitado, definido entre las etiquetas `<saml:Assertion>` y `</saml:Assertion>`. Se puede observar que se trata de aserto de autenticación (`<saml:AuthnStatement>`) y el contexto de autenticación es a través de *password* .

A continuación, procesamos la nueva respuesta obtenida. En caso satisfactorio, se obtiene un objeto XML, que representa el periodo de sesión, con ayuda de la función *restore_profile()*, comentada en 5.2.1. Después, se acepta la SSO y se realizan comprobaciones relativas a los datos de sesión del usuario, con ayuda de la función *dump_profile()*.

Finalmente, se modifican las *cookies* con un identificador del usuario y se le redirige a una página en la que podrá realizar un *Single-Logout*.

5.2.4. Gestión de sesión

La gestión de las sesiones de los usuarios que solicitan los servicios del SP se realiza por medio de un sistema de ficheros, en el cual se almacenan objetos XML que representan los periodos de sesión de aquellos usuarios que hayan completado satisfactoriamente el proceso de SSO. Para acceder al directorio en el que se encuentra almacenada esta información, utilizamos una variable llamada `DIR_SES`, definida en el fichero *session.h*. En este directorio, la información de cada usuario se almacena en un fichero que sigue la siguiente nomenclatura: `sp-name_identifier-session`.

Cada usuario dispone de un fichero que contienen sus datos de sesión. El campo `name_identifier` representa un identificador del usuario de carácter temporal, obtenido de los mensajes intercambiados con el IdP. Para llevar a cabo la lectura y modificación de éste fichero, la librería *IdM* dispone de dos funciones, que son utilizadas durante los procesos de SSO y SLO: *dump_profile()* y *restore_profile()*.

La primera de ellas, se encarga de obtener el identificador del usuario de uno de los campos de la estructura *LassoProfile* y se comprueba si los datos de sesión han sido modificados. En caso afirmativo, se escriben los nuevos datos en un fichero, según la nomenclatura comentada anteriormente. En lo que respecta a la función *restore_profile()*, busca en el directorio si existe un fichero de sesión cuyo

`name_identifier` coincida con el recibido como parámetro de entrada. En caso afirmativo, se lleva a cabo la lectura de este archivo y se asigna su contenido al campo de la estructura *LassoProfile* que representan la información de sesión.

Para realizar las tareas de obtención y modificación de los campos de la estructura `LassoProfile`, nos hemos apoyado en las funciones de Lasso mostradas en la tabla 5.4.

Función	Descripción
`lasso_profile_get_session()`	Obtiene una estructura *LassoSession* de la estructura *LassoProfile*
`lasso_profile_is_session_dirty()`	Comprueba si el campo *session* de *LassoProfile* ha sido modificado
`lasso_profile_set_session_from_dump ()`	Crea una nueva estructura *LassoSession* y la asigna al campo *session* de la estructura *LassoProfile*

Tabla 5.4: Métodos de Lasso utilizados para la configuración de datos de sesión de un usuario

5.2.5. Cierre de sesión

Tras completar con éxito el proceso de SSO, el usuario podrá realizar el cierre único de sesión mediante el perfil de ***Single Logout SOAP***, de acuerdo a las especificaciones de SAMLv2.0. Para ello, la interfaz gráfica de usuario dispone de un botón de *Logout* e invoca a la función *singleLogout()*, definida en la librería desarrollada, en el momento que se detecta que el usuario ha pulsado este botón. Los procesos que se llevan a cabo en esta función consisten en:

1. Obtener de nuevo la configuración del SP.
2. Obtener el identificador del usuario de las *cookies*.
3. Buscar en el directorio de sesiones con ese identificador los datos de sesión e identidad del usuario.
4. Construir una petición de *Single Logout*, en la cual figura la información de sesión encontrada.
5. Enviar la petición por SOAP a la URL especificada en el metadato del IdP donde se encuentra el *SoapEndpoint*, en el cual, éste atiende las solicitudes SOAP que llegan desde el SP.
6. Recibir y procesar la respuesta del IdP. En caso satisfactorio, se actualiza la información de sesión del usuario y se modifican las *cookies* para notificar que el periodo de sesión ha expirado.

La función devuelve como resultado cero, en caso de éxito o un valor negativo en caso de que se haya producido algún error.

Para implementar la funcionalidad de *Single Logout*, hemos utilizado diversos métodos comentados a lo largo de este capítulo, tales como *dump_profile()*, *restore_profile()* y *soap_call()*. También nos hemos apoyado en funciones definidas en la librería Lasso, que nos permiten crear peticiones de *Logout* para enviarlas al IdP, así como procesar sus respuestas. Estas funciones se muestran en la tabla 5.5

Función	**Descripción**
`lasso_logout_init_request()`	Inicializa una nueva petición de *Logout*
`lasso_logout_build_request_msg()`	Construye una petición de *Logout*, modifica los atributos `msg_body` y `msg_url` de la estructura que representa la petición
`lasso_logout_process_response_msg()`	Parsea el mensaje de respuesta y construye un objeto de respuesta de tipo *Logout*

Tabla 5.5: Métodos de Lasso utilizados para realizar *Single Logout* SOAP

Cabe resaltar, que para conseguir que el cierre único de sesión se lleve a cabo a través de SOAP, es necesario el *flag* `LASSO_HTTP_METHOD_SOAP` en la función *lasso_logout_init_request()*.

5.3. Interfaz Gráfica de Usuario

Este módulo está formado por varios programas CGI escritos en C, que principalmente proporciona las siguientes funcionalidades:

- Presenta una interfaz con la que el usuario puede realizar la SSO, gracias a dos botones de *Login* habilitados, y notifica si este proceso se ha completado correctamente. En caso afirmativo, presenta una nueva interfaz con un botón de *Logout* para poder realizar el proceso de SLO y de nuevo, notifica el resultado de este proceso. Para que se puedan llevar a cabo el inicio y el cierre de sesión únicos, se encarga de sincronizar el diálogo SAML entre el SP y el IdP.
- Permite al usuario configurar el metadato del SP.

5.3.1. Estructura

La interfaz gráfica de usuario se compone de un conjunto de ficheros, reflejado en el diagrama de la Figura 5.5, el cual permite proporcionar una idea global de

los métodos que constituyen este módulo.

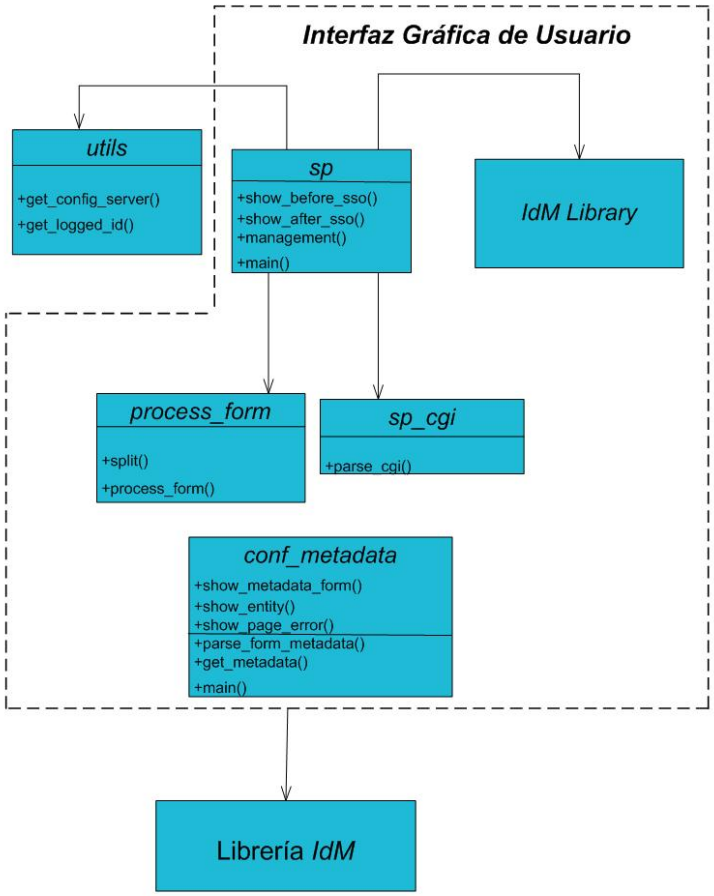

Figura 5.5: Diagrama de ficheros que componen la Interfaz de Usuario y su relación con la librería *IdM*.

La aplicación comienza obteniendo la configuración del SP, para lo cual realiza una llamada a la función *get_config_server()* y obtiene el material criptográfico y ficheros XML necesarios para establecer una comunicación con el IdP. Posteriormente, realiza continuas lecturas de una serie de variables de entorno como la *query string*, el método de petición HTTP y el `Content-Type`. Cuando se obtiene un valor de *query string* distinto de NULL, se realiza un parseo de la misma y se determina qué tipo de acción de las descritas en la sección 5.3.2 se debe realizar.

A lo largo de las siguientes secciones, se describen las nuevas funciones definidas para lograr las funcionalidades descritas y las acciones que debe realizar el programa CGI en función del evento producido.

5.3.2. Interfaz de servicio de prueba

Para llevar a cabo la interacción con el usuario, la aplicación dispone de dos funciones: ***show_before_sso()*** y ***show_after_sso()***. La primera de ellas, muestra al usuario una página sencilla de bienvenida, en la cual se incluye en formulario que éste debe rellenar con una serie de datos personales (nombre, apellidos, DNI, número de teléfono, dirección, etc.). Además, en esta página también se presentan dos botones de *Login* etiquetados con *Login(HTTP-Redirect)* y *Login(HTTP-POST)*, que permiten al usuario elegir el tipo de "binding" con el cual desea realizar la SSO (ver Figura 5.6).

Respecto a la función ***show_after_sso()***, es invocada en caso de que la SSO se haya completado satisfactoriamente (la función *receiveArtifact()*, de la librería *IdM*, lo notifica con un código de éxito) y muestra al usuario una nueva página en la que le felicita por ello (ver Figura 5.7). También proporciona un botón de *Logout*, que permite llevar a cabo el SLO. En caso de que el usuario no se encuentre autenticado en el sistema, se presenta la interfaz de inicio de la Figura 5.6. Cabe destacar, que las interfaces gráficas presentadas al usuario son bastante sencillas, pues el objetivo principal es probar la funcionalidad de la parte de seguridad que proporciona el proveedor de servicios a través de los botones de *Login* y *Logout*, para realizar el inicio y cierre de sesión únicos.

Figura 5.6: Vista de la pantalla que permite al usuario iniciar la SSO.

Figura 5.7: Vista de la pantalla tras completar el proceso de SSO.

Por otro lado, dado que tanto el SP como el IdP se comunican con un usuario agente (que en este caso concreto es un navegador web) para enviar distintos mensajes HTTP/HTTPS de petición o respuesta, por ejemplo el IdP envía asertos de autenticación por referencia al SP a través de la *query string*; es necesario realizar tareas de parseo de ésta y de procesado de formularios HTML.

Esta tareas se llevan a cabo por una función llamada *parse_cgi()*, que obtiene los distintos valores de los parámetros de la *query string* y los almacena en una estructura definida en el mismo fichero, llamada `dataQuery`, que contiene dos campos:

- `op`: Este campo especifica qué acción debemos realizar.
- `artifact`: Este campo almacena el *artifact* enviado por el IdP.

Para procesar los valores rellenados por el usuario en la interfaz de bienvenida, una función llamada *process_form()*, que recibe una cadena de caracteres con el contenido del formulario obtenido de la entrada estándar, y almacena los valores de los diversos campos en una sencilla estructura, llamada `dataUser`, que se muestra a continuación:

```
typedef struct dataUser
{
  char* name;
  char* date;
  char* dni;
```

```
  char* surname;
  char* street_number;
  char* postcode;
  char* city;
  char* province;
  char* telephone_number;
  int binding_select;

} data_user;
```

Como se puede apreciar, además de almacenar los datos personales del usuario, se guarda el "binding" seleccionado para iniciar la SSO.

Para implementar la lógica del programa CGI, se han identificado una serie de posibles acciones que se deben realizar en caso de que se produzcan determinados eventos de interacción con el usuario o encaminados a sincronizar la comunicación entre el SP y el IdP.

Existe un cierto tipo de eventos sobre los que tenemos control directo desde la aplicación CGI y otro tipo de eventos, que podríamos denominar "externos" o producidos por el IdP. Sin embargo, en ambos casos nos basaremos en el valor del primer parámetro de la *query string* para la tomar la decisión acerca de qué acción se debe realizar. Un ejemplo de evento sobre el que tenemos un control directo, es por ejemplo, que el usuario pulse el botón de *Login* .

En el momento que se detecta este hecho, se modifica el valor del primer parámetro de la "*query string*" con la letra "L". En lo que respecta a los eventos "externos", los diferentes valores de los parámetros de la *query string* los hemos añadido en las distintas URLs del metadatato del SP. Para comprender mejor este caso, que puede presentar una mayor complejidad para el lector, vamos a describir un ejemplo concreto.

Si recordamos algunas de las líneas del fichero de metadato del SP:

```
<AssertionConsumerService index="3"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location="https://sp.gast.it.uc3m.es:6443/sp?o=A"/>
```

En la tercera línea se puede apreciar que cuando el IdP envíe una respuesta de autenticación, nos redirigirá a la URL `https://sp.gast.it.uc3m.es:6443/sp?o=A`, por tanto, en este caso el primer parámetro identificado será la letra "A" y nos indicará que la tarea que

se debe realizar es consumir los asertos que llegan desde el IdP. En este caso, además otro de los parámetros que deberemos obtener de la *query string* es la referencia al aserto en cuestión, que presentará un aspecto similar al siguiente:

```
SAMLart=AAMry%2BP%2BeGSDIykD7woghHvkfb%2F%2BdURFMjkwM0QxM
DBENzMwNzU2N0ND
```

Una vez comentados los distintos tipos de eventos que se pueden producir, enumeramos las posibles acciones identificadas y posteriormente, describimos el funcionamiento global de la aplicación CGI:

1. **Acción = A**, se deben recibir y procesar los asertos enviados por el IdP, así como enviarle nuevas peticiones y procesar sus respuestas.
2. **Acción = L**, se ha pulsado uno de los botones de *Login*, por tanto se debe iniciar la SSO enviando una petición de autenticación al IdP.
3. **Acción = N**, se debe mostrar la interfaz de Bienvenida al usuario.
4. **Acción = O**, se ha pulsado el botón de *Logout*, por lo que se debe realizar el proceso de SLO.
5. **Acción = Q**, se ha completado correctamente la SSO y se debe mostrar la interfaz donde se le notifica al usuario este hecho y que le permite hacer un cierre de sesión único. Si el proceso de SSO resultó fallido, se muestra al usuario la interfaz de inicio para que pueda realizar un nuevo intento.
6. **Acción = S**, se debe atender una solicitud SOAP realizada por el IdP.

5.3.3. Interfaz de configuración del metadato del SP

Con el objetivo de ofrecer al administrador del SP la posibilidad de crear el metadato de forma más sencilla, se ha diseñado una interfaz de configuración web. Los datos que éste debe facilitar a la aplicación, son los siguientes:

- El descriptor o nombre de su entidad.
- El certificado que desea utilizar para el propósito de cifrado.
- El certificado que desea utilizar para el propósito de firma.
- Perfiles que desea soportar (SSO, SLO, Federación, etc.).

Para ello, se ha definido la función ***show_metada_form()***, que se encarga de mostrar una página al usuario, que dispone de un formulario para que éste introduzca los datos arriba mencionados. En la Figura 5.8 se muestra un ejemplo del aspecto de esta interfaz de configuración.

Figura 5.8: Vista de la interfaz de configuración del metadato del SP.

Comenzaremos explicando, cómo se ha implementado la lógica relativa a la obtención de los datos del formulario rellenado por el usuario a través de la interfaz web en *conf_metadata*. Posteriormente describiremos las funciones que intervienen en la parte de creación del metadato.

5.3.3.1. Obtención de los datos del formulario de configuración

El tipo de contenido que se ha utilizado para codificar el conjunto de datos del formulario para su envío al servidor es *multipart/form-data*, ya que en este formulario se deben enviar los ficheros que contienen los certificados para cifrar y firmar. Para almacenar las preferencias seleccionadas por el usuario, hemos definido la estructura que se detalla a continuación:

```
 typedef struct
{
   char* entity_descriptor;
```

```
char* encryption_certificate;
char* encryption_certificate_name;
int encryption_certificate_length;
char* signing_certificate;
char* signing_certificate_name;
int signing_certificate_length;
int  type;
int name_metadata_file;

} metadata_conf;
```

- El campo *entity_descriptor* contiene la URL de la entidad que deseamos configurar. Esta URL será parseada con la ayuda de la función *url_parse()* para obtener a partir del *path* el nombre del fichero en el que almacenaremos el metadato y posteriormente asignaremos al campo *name_metadata_file*.

- Otros campos de interés, son aquellos que nos permiten almacenar el contenido, el nombre y la longitud de los ficheros proporcionados por el usuario, y que contienen los certificados para los propósitos de cifrado y firma: *encryption_certificate*, *encryption_certificate_name*, *encryption_certificate_length*, *signing_certificate*, *signing_certificate_name* y *signing_certificate_length*.

- Por último, el campo *type* nos indica el tipo de metadato seleccionado por el usuario. Cabe resaltar, que en este proyecto sólo se ha contemplado un tipo de metadato genérico que define los perfiles de SSO y SLO con el uso de distintos "bindings", pero se contempla como trabajo futuro proporcionar al usuario la posibilidad de configurar un metadato que soporte más tipos de perfiles definidos en SAML.

A continuación, hemos definido el método ***parse_form_metadata()***, que recibe como parámetro de entrada un puntero a esta estructura y se encarga de obtener el valor de los distintos campos del formulario, con la ayuda de funciones de la librería *qDecoder* y asignarlos a la estructura `metadata_conf`.

5.3.3.2. Generación del metadato del SP

La función ***parse_form_metadata()*** es la encargada de generar la estructura de árbol XML que define un metadato SAML, similar al mostrado en la sección 5.2.2. Para ello, recibe como parámetro de entrada una estructura de tipo

`metadata_conf`. En esta función se realizan comprobaciones acerca de los datos introducidos. Se verifica que el usuario ha rellenado en el formulario la información solicitada y que los ficheros que contienen los certificados sigan el formato PEM. A continuación, se procede a la creación del árbol XML y para ello, se debe introducir el contenido de los certificados para cifrar y firmar, facilitados por el usuario, en el momento que lleguemos a la parte de generación de los nodos `<KeyDescriptor>`, cuando el atributo "*use*" tome los valores "'*encryption*" y "*signing*", respectivamente. Si el resultado de estas operaciones es exitoso, la estructura del árbol XML que representa el metadato del SP, se almacena en un fichero, en el directorio *htdocs* de Apache y se devuelve como resultado cero. En caso contrario, se devuelve un código de error negativo, que indica el tipo de error que se ha producido (por ejemplo, que los certificados no siguen el formato PEM, que falta introducir algún dato del formulario rellenado por el usuario, etc).

Si el metadato se ha podido crear correctamente, la aplicación mostrará una página, con ayuda de la función *show_entity()*, que dispone de un enlace para que el usuario pueda visualizar el metadato generado. En caso contrario, se mostrará una página de error.

Capítulo 6

Pruebas

En este capítulo se presentan las pruebas realizadas y los resultados obtenidos, en primer lugar para el proveedor de servicios (SP) de ZXID y los servicios ofrecidos por esta herramienta, utilizada para proporcionar su soporte. También se presentan las pruebas llevadas a cabo para probar el funcionamiento del proveedor de identidad (IdP) y pruebas de integración de éste y el SP basado en ZXID. Después se presentan las pruebas realizadas para comprobar el funcionamiento del proveedor de servicios implementado, así como su interoperabilidad con el IdP. Finalmente, se presentan pruebas de interoperabilidad realizadas entre los dos SPs de nuestra plataforma.

Las distintas pruebas realizadas, se recogen en tablas resumen a lo largo de este capítulo. Se ha asignado un código para hacer referencia cada prueba, formado por 3 o 4 dígitos, en función de la siguiente notación:

- Cada prueba puede pertenecer a la sección x del presente capítulo. Por que se denotará como *6x*.

- Cada prueba puede pertenecer a la subsección y del presente capítulo. Por que se denotará como *6xy*. En caso de que no exista tal subsección, el código de la prueba contendrá tres dígitos en lugar de cuatro.

- El número de la prueba realizada para la sección *6x* o la subsección *6xy* se denotará con el dígito z. Por lo que cada prueba se denotará con el código *6xyz* ó *6xz*.

6.1. Pruebas de módulos individuales

Las primeras pruebas consistieron en verificar la funcionalidad por separado de los servicios instalados. Estas pruebas fueron bastante sencillas puesto que para ver la funcionalidad completa de estos servicios era necesario integrarlos, configurando las relaciones entre ellos. Por tanto, en esta sección explicamos en qué consistieron las primeras pruebas.

6.1.1. Pruebas de los servicios de ZXID

Una vez instalado ZXID y todas sus dependencias las primeras pruebas consistieron en lanzar los servicios que éste incluye desde distintos navegadores. Los servicios de prueba que soporta esta librería son:

- **Servicio básico "HelloWorld"**, muestra una interfaz gráfica sencilla que permite añadir la URL del metadato del IdP y poder realizar el proceso de SSO. En esta interfaz gráfica se ofrece la posibilidad de utilizar por defecto un IdP de *Orange* o un IdP llamado *idp.symdemo.com*.
- **Servicio Web "HelloWorld"**, ofrece una interfaz similar al servicio básico de "HelloWorld". Permite realizar SSO entre un WSP y un IdP. Ofrece proveedores de identidad por defecto para realizar el inicio de sesión único y también permite introducir la URL del metadato de un IdP que escojamos.
- **Servicios web que siguen la sintaxis HRXML**[1]. El objetivo de HR-XML es representar documentos relacionados con la gestión de Recursos Humanos, como por ejemplo currículos vitae, la información de nómina de sueldos, beneficios de matricula, etc de forma estructurada. Los servicios que ofrece ZXID disponen de un proveedor de servicios web (WSP) y un cliente de servicios web (WSC), que permiten intercambiar este tipo de datos a través de documentos XML.

Las pruebas realizadas acerca de los servicios de prueba de ZXID se recogen en la tabla 6.1

[1]HRXML (*Human Resources Extensible Markup Language*) es una biblioteca desarrollada por el HR-XML Consortium Inc. para apoyar una variedad de procesos de negocio relacionados con la gestión de los Recursos Humanos.

Prueba realizada	Código	Resultado	Observaciones
Acceso al Servicio básico "HelloWorld".	6111	OK	La página de este servicio se carga correctamente en el navegador. Dispone de botones que permiten hacer SSO con varios IdPs que ofrece por defecto ZXID o con un IdP que configuremos nosotros. La interfaz de este servicio también ofrece un enlace para acceder a la URL del metadato del SP.
Acceso a la URL que contiene el metadato del SP	6112	OK	En un primer momento, al tratar de visualizar con el navegador el metadato del SP utilizando la URL, nos mostraba una página en blanco. El problema se debía a un mal funcionamiento de la función `write_all_fd` de ZXID. Para solucionar el problema, realizamos una serie de modificaciones en los ficheros `zxid.c` y `zxidmeta.c`.
Pruebas de SSO entre el SP del servicio básico e IdPs que ofrece por defecto la interfaz gráfica de "HelloWorld"	6113	NO OK	Cuando tratamos de iniciar la SSO con los dos IdPs que se ofrecen por defecto en la interfaz gráfica, se nos informa de que estos servidores no se encuentran disponibles, por lo que es necesario disponer de nuestro propio IdP. En la documentación se recomienda *Authentic* como IdP interoperable.
Acceso al Servicio Web "HelloWorld"	6114	NO OK	La interfaz gráfica se puede cargar en el navegador, pero éste requiere un servidor de descubrimiento para hacer SSO.
Acceso a los Servicios web que siguen la sintaxis HRXML	6115	NO OK	Para probar este escenario, es necesario un servidor de descubrimiento. Por tanto, el resultado de esta prueba fue exitoso respecto a la carga del cliente (WSC), pero el servidor daba un error interno.

Tabla 6.1: Pruebas de acceso a distintos servicios de ZXID.

En la tabla 6.1, se puede observar que en la prueba que hemos denotado con el código *6112*, fue necesario hacer modificaciones en el código fuente de los ficheros `zxid.c` y `zxidmeta.c`, para poder visualizar correctamente el metadato del SP en el navegador. Por tanto, en el fichero `zxid.c`, en la línea 572 hemos sustituido la llamada a la función `write_all_fd`, que escribe el contenido de un fichero por la salida estándar, por una llamada a la función `printf` de la siguiente manera:

```
case 'B':  /* Metadata */

 //write_all_fd(1, "Content-Type: text/xml\r\n\r\n", \
 // sizeof("Content-Type: text/xml\r\n\r\n")-1);
printf("Content-Type: text/xml\r\n\r\n");
return zxid_send_sp_meta(cf, &cgi);
```

Esta misma modificación fue necesaria hacerla en el fichero `zxidmeta.c` en la función llamada `zxid_send_sp_meta()` de la siguiente manera:

```
/* Called by:  main, opt */
int zxid_send_sp_meta(struct zxid_conf* cf, struct zxid_cgi* cgi)
{
  struct zx_str* ss = zxid_sp_meta(cf, cgi);
  if (!ss)
    return 0;
//write_all_fd(stdout, ss->s, ss->len);
printf(ss->s);
zx_str_free(cf->ctx, ss);
  return 0;
}
```

Después de realizar estos cambios, pudimos visualizar correctamente la página que contiene el metadato del SP en el navegador.

Por otra parte, tras realizar el conjunto de pruebas presentadas en la tabla 6.1, pudimos extraer las siguientes conclusiones:

- Para poder probar el funcionamiento de los perfiles SSO y SLO era necesario disponer de nuestro propio IdP y realizar la integración entre el SP y el IdP, dado que los servicios de los proveedores de identidad de prueba, que se ofrecen en la interfaz del SP no se encuentran disponibles.
- Los servicios web de "HelloWorld" y el servicio basado en la sintaxis HRXML utilizan un servidor LDAP que se debe encontrar a partir de un servidor de descubrimiento. El servidor de descubrimiento no está disponible como código abierto, sino como un producto comercial, por tanto, las pruebas de estos servicios no se pudieron concluir con éxito. Este aspecto lo comentaremos en mayor profundidad en el capítulo 7.

6.1.2. Pruebas del Proveedor de Identidad (IdP)

Una vez instalado el IdP con todas sus dependencias y librería de soporte, y exitosamente configurado, se realizaron pruebas de los servicios ofrecidos por el IdP. Estas pruebas incluyen la configuración del IdP a través de la interfaz web facilitada por *Authentic* y la creación de distintos usuarios de prueba. Cabe destacar, que en la primera fase del proyecto la gestión de usuarios se realizaba

directamente en el sistema de ficheros. Las pruebas más relevantes realizadas para el IdP se recogen en la tabla 6.2.

Prueba realizada	Código	Resultado	Observaciones
Configuración del proveedor de identidad desde la interfaz de administración de *Authentic*. Para ello, facilitamos a la aplicación los ficheros que contienen la clave privada y el certificado del IdP	6121	OK	Tras realizar esta operación, se generan dos metadatos para el IdP. El primero sigue las especificaciones de SAML y el segundo las de Liberty. En nuestro caso, trabajaremos con el metadato de SAML. Visualizamos con el navegador que las páginas de los metadatos se cargan correctamente.
Creación de un usuario sin rol de administrador desde la interfaz de administración de *Authentic*	6122	OK	Si accedemos al enlace `Admin→Identity Management` podemos observar la información relativa al usuario generado. Esta información se almacena en un sistema de ficheros en el IdP. Si tratamos de acceder a la interfaz de administración con este usuario obtenemos un *Forbidden* en el servidor, que es lo correcto puesto que no dispone de permisos de administración.
Creación de un usuario con rol de administrador desde la interfaz de administración de *Authentic*	6123	OK	Si accedemos al enlace `Admin→Identity Management` podemos observar la información relativa al usuario generado. Esta información se almacena en un sistema de ficheros en el IdP. Con este usuario podremos acceder a la interfaz de administración del IdP.
Autenticación de usuarios que se encuentran registrados en el IdP.	6124	OK	Se realiza una autenticación correcta de los usuarios registrados con rol de administrador y de aquellos que no disponen de privilegios de administración.
Autenticación de usuarios que no se encuentran registrados en el IdP.	6125	OK	La autenticación falla y el IdP lo notifica con un mensaje de error.

Tabla 6.2: Pruebas del proveedor de identidad.

6.1.3. Pruebas del Servidor LDAP

Para llevar a cabo una gestión de usuarios eficiente en nuestro sistema de gestión de identidad, instalamos y configuramos un servidor LDAP. Después realizamos pruebas de inserción, borrado y búsqueda de usuarios en este directorio, a través de la consola y del explorador de directorios proporcionado por la herramienta *phpLDAPadmin*. En la tabla 6.3 se reflejan las pruebas realizadas más relevantes.

Prueba realizada	Código	Resultado	Observaciones
Creación de cuentas de usuarios de tipo *posixAccount* e *inetOrgPerson* a través de la herramienta de línea de comandos *ldapadd*	6131	OK	La cuentas de usuario se crean correctamente. Si buscamos uno de los usuarios creados en el directorio, mediante el comando *ldapsearch*, LDAP nos facilita la información del usuario solicitado.
Modificación de usuarios a través de la herramienta de comandos *ldapmodify*	6132	OK	La información de los usuarios sólo puede ser modificada por el administrador del directorio LDAP.
Creación de cuentas de usuarios de tipo *posixAccount* e *inetOrgPerson* a través de la herramienta *phpLDAPadmin*	6133	OK	Las cuentas de usuario se pueden crear rellenando el valor de los atributos de esta clase de objetos a través de un formulario proporcionado, por la interfaz web de *phpLDAPadmin*; o importando un fichero en formato LDIF.
Creación de cuentas de usuarios de tipo *userAuthentic*	6134	OK	Las cuentas de usuario se pueden crear importando un fichero en formato LDIF o rellenando el valor de los atributos de esta nueva clase de objeto a través de un formulario. Hemos realizado una configuración en *phpLDAPadmin* para que se puedan crear este nuevo tipo de cuentas utilizando la segunda opción, al igual que se ofrece esta posibilidad para crear el resto de clases de objeto definidas en LDAP.
Creación de un usuario con rol de administrador	6135	OK	El usuario se crea correctamente.
Búsqueda de usuarios a través de *phpLDAPadmin*	6136	OK	La herramienta permite realizar búsquedas en el árbol completo o en los niveles que se especifiquen. Se ofrecen filtros de búsqueda y también se permite elegir qué atributos se deben mostrar del usuario solicitado.
Modificación y borrado de usuarios a través *phpLDAPadmin*	6137	OK	Estas modificaciones sólo las pueden realizar aquellos usuarios administradores de LDAP que dispongan de permisos.

Tabla 6.3: Pruebas del servidor LDAP.

6.2. Pruebas de integración e interoperabilidad

En este conjunto de pruebas, se llevaron a cabo pruebas de integración e interoperabilidad entre los distintos proveedores para verificar la correcta ejecución de los perfiles SSO y SLO de SAML. Las pruebas de integración consisten básicamente en establecer una relación de confianza mutua entre el SP y el IdP de forma manual a través de metadatos; por tanto, primero se declara ZXID como SP de *Authentic* y segundo se configura *Authentic* como el IdP de ZXID. La segunda

parte, son la pruebas de interoperabilidad para llevar a cabo el proceso completo de registro y "des-registro" del servicio.

Por otro lado, en una fase posterior de desarrollo, se estimó oportuno realizar la gestión de usuarios en el IdP a través de un servicio de directorio LDAP, con el fin de que esta tarea fuera más eficiente. Por lo que fue necesario realizar nuevas pruebas de integración del IdP con el servidor LDAP. Estas pruebas también se recogen en esta sección.

6.2.1. Pruebas de integración del SP de ZXID con el IdP

En el capítulo 4 realizamos una descripción acerca del procedimiento para establecer una relación de confianza entre el SP y el IdP. En esta sección vamos a tratar este tema con mayor nivel de detalle. Recordemos que para hacer la declaración de ZXID como SP de *Authentic* existen dos formas, una es pasar directamente el certificado e información relacionada con el SP y la otra es introducir simplemente la URL con el metadato. Durante el proyecto se han realizado pruebas con las dos opciones. Para utilizar la segunda opción, por ejemplo, como indicamos también en 4.4.1, es necesario seleccionar en la interfaz del IdP:

`Settings→LibertyProviders→Create new from remote URL`
y especificar la URL que contiene el metadato del SP, que es:

```
https: //sp.gast.it.uc3m.es:8443/zxid?o=B
```

Para hacer la declaración en el sentido contrario, es decir, establecer *Authentic* como IdP de ZXID, se sigue un procedimiento similar y para ello, se indica la URL del metadato del IdP en la interfaz del SP de ZXID:

```
https://idp.gast.it.uc3m.es:5443/saml/metadata
```

En la tabla 6.4 reflejamos los resultados obtenidos para establecimiento de una relación de confianza entre los dos proveedores, que en un primer momento fueron fallidos, y explicamos los motivos por los que se producían errores.

Prueba realizada	Código	Resultado	Observaciones
Declaración de ZXID como SP de *Authentic*, pasando la URL del metadato del SP al IdP	6211	OK	En un primer momento esta prueba resultó fallida por los siguientes motivos: 1. El IdP no podía procesar correctamente el metadato del SP porque no se reconoce uno de los *bindings* definidos en éste. Dicho *binding* se denomina `HTTP-POST-SimpleSign` y como indicamos en 4.4.1, es un *draft*, por lo que todavía no está incorporado y por este motivo, puede ser que todavía no esté implementado en *Authentic*. 2. El IdP no llevaba a cabo la correcta comprobación de uno de sus objetos, mostrando el siguiente error: `Attribute Error:'Provider' object has no atribute 'this'.` Tras solucionar estos problemas, observamos que el metadato se procesa adecuadamente en el IdP y que el SP se añade su lista de proveedores conocidos. El metadato del SP queda almacenado en un directorio del IdP.
Declaración de ZXID como SP de *Authentic*, pasando el certificado y la cadena de certificados del SP, asicomo su metadato al IdP	6212	OK	Esta prueba también se realizó tras solucionar los problemas de 6211. La declaración es exitosa. Se almacena el material criptográfico y el metadato en un directorio del IdP.
Declaración *Authentic* como IdP de ZXID	6213	OK	El SP procesa adecuadamente el metadato del IdP y lo almacena en su "caché de metadatos".

Tabla 6.4: Pruebas de establecimiento de una relación de confianza mutua entre el SP y el IdP.

- La solución al primer problema que se produce en la prueba denotada en la tabla 6.4 como *6211*, ha consistido en modificar el código de ZXID para que no aparezca definido el *binding* `HTTP-POST-SimpleSign` en el metadato del SP. Estas modificaciones se han realizado en el fichero `zxidmeta.c`, comentando la parte de código correspondiente a las líneas 503-505:

```
za = zxid_ac_desc(cf, SAML2_POST_SIMPLE_SIGN, "?o=P", "5");
za->gg.g.n = &sp_ssod->AssertionConsumerService->gg.g;
sp_ssod->AssertionConsumerService = za;
```

- El segundo error comentado en la prueba *6211*, se produce en uno de los

ficheros del IdP, llamado `settings.ptl`, concretamente en la línea 195, donde se realizaba la siguiente comprobación:

```
if p.this is None:
```

Donde el objeto `p` representa un proveedor. Para solucionar este error fue necesario comprobar simplemente si el proveedor era nulo, en lugar de preguntar por un atributo concreto:

```
if p is None:
```

De esta manera, al realizar el *submit* ya no se producen errores y se realiza correctamente la declaración de ZXID como SP a *Authentic*.

6.2.2. Pruebas de interoperabilidad entre el SP de ZXID y el IdP

Una vez establecida la relación de confianza mutua entre los dos proveedores, podemos probar el funcionamiento de distintas variantes de los perfiles SSO y SLO (como medida de interoperabilidad entre proveedores), a través del uso de diferentes "bindings". Estas pruebas se realizaron desde navegadores de dispositivos móviles y máquinas remotas a los servidores. En concreto, se utilizaron los siguientes navegadores desde máquinas con sistemas operativos Windows y Linux: *Firefox*, *Internet Explorer* y *Konqueror*. También se utilizaron los siguientes modelos de terminales móviles para llevar a cabo las pruebas: Nokia N96 (S60 OSS *Browser*), *Qtek* S200 y Nokia 770 *Internet Tablet*.

Para ello, si accedemos a la interfaz gráfica del SP a través de la siguiente URL: `https://sp.gast.it.uc3m.es:8443/zxid`, tenemos la posibilidad de pulsar los botones *Login(A2)*, *Login(P2)*, *Login(any)* y *Login (S2)*. En función del botón que pulsemos, se efectuará SSO siguiendo distintos *bindings*, como puede verse en el capítulo 4.

Las pruebas se llevaron a cabo con los tres primeros botones, debido a que el botón *Login(S2)* implementa un *binding* (`HTTP-POST Simple Sign`), que se trata de un *draft* y de momento no está soportado por *Authentic*. En un principio, los resultados fueron erróneos. Los problemas fueron de nuevo ocasionados por incompatibilidad en los metadatos, reflejados en las pruebas con códigos *6221* y *6222* de la tabla 6.5.

La solución a estos problemas se comentó en 4.4.1 y recordemos, consistió en modificar la interfaz de configuración del *Authentic* para introducir el certificado

en lugar de la clave pública. Así, el proceso de generación del metadato de SAML se modificó para que cargase un certificado en vez de una clave pública. Esta modificación la hemos realizado en el fichero *settings.ptl*, que se encuentra en el directorio de administración del *Authentic*:

```
Linea 1395:
   idp_key = ''
        if os.system('openssl version > /dev/null 2>&1')==0:
         # use system calls for openssl since PyOpenSSL
         # doesn't expose the necessary functions.
         dir = get_publisher().app_dir
         publickey_fn = os.path.join(dir, 'public-key.pem')
   der_key = os.popen('openssl x509 -in %s -outform der'
                      % publickey_fn).read()
```

De esta forma, no se modificó la creación del metadato de acuerdo con ID-FF de Liberty Alliance.

Tras solucionar estos problemas, se seleccionó el IdP previamente configurado, utilizando los botones descritos del interfaz del SP. Dichos botones redirigen al usuario a la página de autenticación del *Authentic*, donde se utilizaron usuarios de prueba creados inicialmente para autenticarse.

Si el resultado de la autenticación ha sido exitoso, entonces automáticamente se redirige al usuario a la página principal del servicio de ZXID. En esta página se le indica la correcta autenticación y se le ofrece la posibilidad de realizar un cierre de sesión único de diversas maneras, mediante cinco posibles botones (ver Figura 4.3).

En esta parte de las pruebas sólo utilizamos las tres primeras opciones de cierre de sesión descritas en 4.2.1, puesto que en el metadato de SAML del IdP no está contemplado el caso de uso de terminación de federación.

Cabe destacar, que al utilizar el botón para cerrar la sesión localmente (*Local Logout*), el resultado fue exitoso, mientras que al tratar de cerrar sesión por medio de los botones *Single Logout (Redir)* y *Single Logout (SOAP)*, se producía un error interno en el servidor del SP.

El error se debía a que ZXID requiere que el metadato del IdP contenga certificado(s) con propósitos de cifrado y de firma. El metadato del *Authentic* únicamente contiene el certificado con el propósito de la firma, que fue él que se configuró inicialmente.

Así, la solución consistió en modificar el código del IdP para que incluyese en el metadato también el propósito de cifrado para el mismo certificado, puesto

que está incluido dentro de los propósitos de uso del mismo. La modificación consistió en añadir el siguiente fragmento de código a partir de la línea 1407 del fichero `settings-ptl`:

```
idp_key2 = """<KeyDescriptor use="encryption">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2001/04/xmlenc#">
 <ds:X509Data>
   <ds:X509Certificate>%s</ds:X509Certificate>
     </ds:X509Data>
   </ds:KeyInfo>
</KeyDescriptor>""" % base64.encodestring(der_key)
```

Y en modificar la línea 1440 por la siguiente:

```
return '\n'.join([prologue, idp_head, idp_key, idp_key2,
 idp_body, epilogue])
```

En la tabla 6.5 se recogen los resultados de las pruebas realizadas desde navegadores de máquinas remotas.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Prueba del perfil **SSO Web iniciado por el SP: POST-*Artifact Bindings***	6221	OK	El metadato del IdP contiene únicamente la clave pública y no el certificado completo tal como se define en las especificaciones de SAML. Por lo que se producía un error interno en el servidor del SP. Tras solucionar este problema, se completa satisfactoriamente la SSO. Necesario borrar las *cookies* del navegador en caso de que se desee probar de nuevo el proceso de inicio de sesión único.
Prueba del perfil **SSO Web iniciado por el SP: *Redirect*-POST**.	6222	OK	De nuevo, sucede el mismo problema que en 6221, es decir, el metadato del IdP contiene únicamente la clave pública, por lo que se produce un error interno en el servidor del SP. Tras solucionar este problema, se completa la SSO. Necesario borrar las *cookies* del navegador en caso de que se desee probar de nuevo el proceso de inicio de sesión único.

Prueba del proceso de SSO utilizando el botón *Login(any)* de la interfaz gráfica del SP.	6223	NO OK	Se produce un error en el servidor del IdP. Este error se produce en los distintos navegadores utilizados (*Firefox*, *Internet Explorer* y *Konqueror*). En la documentación de ZXID no se facilita información acerca de la funcionalidad de este botón. Si consultamos la documentación del IdP (*Authentic*), encontramos una tabla de interoperabilidad [2] con ZXID, en la que tampoco se reflejan pruebas realizadas con este mecanismo. Por lo que esta prueba no se ha podido concluir con éxito.
Prueba del perfil **SSO Web iniciado por el SP: POST-*Artifact Bindings*** utilizando un usuario no registrado en el servidor LDAP que utiliza el IdP para autenticar a los usuarios del sistemas.	6224	OK	Al iniciar la SSO desde el SP, somos redirigidos al IdP. Al introducir las credenciales incorrectas, el proveedor de identidad notifica que la autenticación ha fallado.
Prueba del perfil de SLO a través del botón de *Logout(Redirect)* tras haber completado el proceso de SSO Web con distintos "bindings".	6225	OK*	Se lleva a cabo el cierre de sesión único, pero el servicio del SP nos redirige a una página del IdP, en la que se muestra un mensaje de "Page Not Found". Este problema sucede con todos los navegadores que hemos utilizado (*Firefox*, *Internet Explorer* y *Konqueror*).
Prueba del perfil de SLO a través del botón de *Logout(SOAP)* tras haber completado el proceso de SSO Web con distintos "bindings".	6226	OK*	En ocasiones el servidor del SP tarda bastante en llevar a cabo este proceso y se queda "colgado". Este problema ocurre con los distintos navegadores que hemos utilizado (*Firefox*, *Internet Explorer* y *Konqueror*).
Prueba del perfil de SLO a través del botón de *Local Logout* tras haber completado el proceso de SSO Web con distintos "bindings".	6227	OK	Los resultados son similares si empleamos *Firefox*, *Internet Explorer* y *Konqueror*.

Tabla 6.5: Pruebas de los perfiles de SSO y SLO.

También realizamos pruebas utilizando como clientes los navegadores de distintos dispositivos móviles, mencionados al principio de esta sección. De estas pruebas, podemos que concluir que los problemas que surgieron inicialmente al integrar el SP y el IdP (ver pruebas *6221* y *6222*), se producían independientemente del navegador utilizado. Del mismo modo, se producía el error comentado en la prueba *6223*, al tratar de hacer la SSO con el botón *Login(any)*. En lo que se

[2] http://lasso.entrouvert.org/documentation/interoperability

refiere a las pruebas de los procesos de inicio y cierre de sesión únicos utilizando distintos "bindings", también resultaron exitosas. Por lo que podemos concluir, que los navegadores empleados soportan las cabeceras de las tramas HTTPS, intercambiadas en el diálogo SAML entre el SP y el IdP, durante los procesos de SSO y SLO.

6.2.3. Pruebas de integración del IdP y el servicio de directorio LDAP

Las pruebas más relevantes realizadas durante el proceso de integración del proveedor de identidad con el servidor LDAP se reflejan en la tabla 6.6.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Configuración de los parámetros del servidor LDAP a través de la interfaz de administración del IdP.	6231	OK	Se ha modificado la opción por defecto de gestión usuarios a través de ficheros. El IdP es capaz de comunicarse con el servidor LDAP.
Creación de cuentas de usuarios de tipo *posixAccount* e *inetOrgPerson* desde la interfaz de configuración del IdP.	6232	NO OK	Se produce un error en el IdP en el que se indica, que es necesario que las clases de objeto utilizadas para crear las cuentas de usuario deben contener el atributo obligatorio *password*. En los esquemas de las clases de objeto *posixAccount* e *inetOrgPerson* se contempla que este atributo es opcional.
Añadir el nuevo tipo de cuenta *userAuthentic* en la configuración del servidor LDAP.	6233	OK	Se pueden crear nuevos objetos en LDAP utilizando la clase *userAuthentic*.
Creación de cuentas de usuarios de tipo *userAuthentic* desde la interfaz de administración del IdP.	6234	NO OK	Se produce un error en el IdP en el que se indica que no es capaz de generar la estructura de árbol LDAP.
Creación de cuentas de usuarios de tipo *userAuthentic* desde el explorador de directorio *phpLDAPadmin*.	6235	OK	El IdP es capaz de hacer una lectura correcta de la información almacenada en el servicio de directorio. Se ha creado un usuario administrador y usuarios sin permisos de administración.
Modificación y borrado de usuarios a través *phpLDAPadmin*.	6236	OK	Estas modificaciones se llevan a cabo por un usuario con perfil de administrador LDAP. Al llevar a cabo la lectura de la información almacenada en el servidor LDAP, desde el IdP, se reflejan las actualizaciones realizadas.
Autenticación de un usuario con rol de administrador desde la interfaz de Login del IdP.	6237	OK	El IdP lleva a cabo la autenticación correctamente con la ayuda del servidor LDAP.

Autenticación de un usuario sin rol de administrador desde la interfaz de Login del IdP.	6238	OK	El IdP lleva a cabo la autenticación correctamente con la ayuda del servidor LDAP.
Autenticación de un usuario no registrado en LDAP desde la interfaz de Login del IdP.	6239	OK	El IdP notifica que la autenticación ha sido fallida.

Tabla 6.6: Pruebas de integración del IdP con el servidor LDAP.

En la tabla 6.6, se puede apreciar que el IdP soporta el modo lectura de los datos almacenados en el servidor LDAP, sin embargo, la creación de usuarios a través de la interfaz de *Authentic* resultó fallida (pruebas *6232* y *6234*). Al consultar con sus desarrolladores, nos informaron de que el modo escritura en LDAP a través de *Authentic* todavía no es muy estable. La creación de usuarios desde la interfaz del IdP, se puede realizar cuando la gestión de usuarios se realiza a través de sistemas de ficheros, pero al utilizar el servicio de directorio, estos se deben crear a través de LDAP. Para ello, utilizamos el explorador de directorio LDAP proporcionado por la herramienta *phpLDAPadmin*.

6.3. Pruebas del Proveedor de Servicios implementado

En esta sección explicamos el conjunto de pruebas realizado para comprobar el funcionamiento del SP desarrollado en el capítulo 5 y su interoperabilidad con el IdP.

Durante la realización de estas pruebas fue necesario generar certificados digitales para realizar la configuración del SP. También se definieron ficheros de metadatos de prueba y fue necesario contar con el metadato del IdP, así como su certificado y cadena de certificados. Para llevar a cabo estas pruebas hemos utilizado también diversas herramientas *software* como: *WireShark*, *IEWatch* y *Valgrind*. Por lo que explicamos brevemente en qué consiste cada una y para qué se ha utilizado.

- *WireShark* es un analizador de protocolos, de *software* libre y se ejecuta sobre la mayoría de los sistemas operativos Unix y compatibles. Permite realizar análisis y solucionar problemas en redes de comunicaciones. En este proyecto se ha utilizado esta herramienta para visualizar las tramas

intercambiadas entre el SP y el IdP. Sin embargo, debido a que el contenido de éstas se encuentra cifrado, con *WireShark* únicamente podíamos verificar aspectos tales como, que las direcciones IP de origen y destino de las tramas eran las adecuadas, lo cual nos indicaba que el diálogo entre los proveedores se estaba llevando a cabo, pero no podíamos observar el contenido en claro de los mensajes intercambiados.

- Para visualizar el contenido de las tramas intercambiadas por HTTPS, se ha utilizado la herramienta *IEWatch*, que es un *plug-in* para *Microsoft Internet Explorer*.
- En lo que respecta a *Valgrind*, es un conjunto de herramientas de *software* libre que ayuda en la depuración de problemas de memoria y rendimiento de programas.

Además, cabe destacar que para realizar las pruebas presentadas en esta sección se han empleado los navegadores web: *Firefox*, *Internet Explorer* y *Konqueror*.

6.3.1. Pruebas de la librería *IdM*

Inicialmente, realizamos un conjunto de pruebas para verificar que los distintos pasos del diálogo llevado a cabo entre el SP y el IdP para realizar los distintos perfiles implementados, se llevaban a cabo correctamente.

Recordemos, que el perfil de SSO POST-*Artifact Bindings* se puede llevar a cabo con este proveedor utilizando desde el SP los *bindings* `HTTP-Redirect` y `HTTP-POST`, por lo que inicialmente se realizó un programa independiente de prueba que realizaba la secuencia completa de mensajes con el *binding* `HTTP-Redirect` y posteriormente, se siguió el mismo procedimiento con el *binding* `HTTP-POST`. En la tabla 6.7 se recogen las pruebas realizadas más relevantes.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Generación del mensaje de petición de autenticación desde el SP utilizando el "binding"`HTTP-Redirect`	6311	OK	Los mensajes se generaban correctamente, el código devuelto por las funciones de *IdM Library* utilizadas para este fin es exitoso. El IdP recibe correctamente esta petición y nos envía un mensaje de redirección a su página de *Login*.
Generación del mensaje de petición de autenticación desde el SP utilizando el "binding"`HTTP-POST`.	6312	OK	Los mensajes se generaban correctamente, el código devuelto por las funciones de *IdM Library* utilizadas para este fin es exitoso. El IdP recibe correctamente esta petición y nos envía un mensaje de redirección a su página de *Login*.

Generación de la URL a la que nos debe redirigir el SP cuando envía la petición de autenticación con el "binding" `HTTP-Redirect`.	6313	OK	Esta URL es donde el IdP recibe dicha petición, especificada su metadato, acompañada de la *query string*, generada en el proceso de petición de autenticación. Automáticamente el IdP nos redirige a su interfaz de Login para introducir los datos del usuario.
Recepción del *artifact* que el IdP envía como respuesta tras introducir los datos de un usuario de prueba en su interfaz de *Login*.	6314	OK	El SP procesa la referencia al aserto (*artifact*) correctamente y envía una solicitud SOAP al IdP para obtener el valor del aserto.
Envío del mensaje SOAP desde el SP para obtener el valor del aserto emitido por el IdP.	6315	OK	El IdP procesa el mensaje adecuadamente. Realiza una comprobación exitosa de la firma XML contenida en el mensaje SOAP.
Recepción del mensaje SOAP enviado por el IdP.	6316	OK	El mensaje contiene un código de éxito y el valor de un aserto de autenticación. Sin embargo, tuvimos algunos problemas al procesar este mensaje, debido a que la lectura del mismo se realizaba de forma incompleta. El error se producía porque en una de las funciones de la librería *LibNeon*, que se encargaba de la lectura de los bloques de la respuesta de una petición POST, había una condición incorrecta que estaba provocando la interrupción en la lectura del mensaje de respuesta, sin que éste se hubiera leído completamente. Tras modificar esa condición errónea, se consigue leer toda la respuesta, el SP la procesa satisfactoriamente y finalmente, se completa la SSO.

Tabla 6.7: Pruebas de desarrollo unitario del SP.

6.3.2. Pruebas de integración entre el SP y el IdP

En este conjunto de pruebas, se llevaron a cabo pruebas de integración entre los dos proveedores. Las pruebas de integración consisten, en establecer una relación de confianza mutua entre el SP y el IdP de forma manual a través de metadatos. En la tabla 6.8 se recogen las pruebas más relevantes, realizadas para la integración de los proveedores.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Declaración del SP implementado como proveedor de servicios de *Authentic*	6321	OK	El IdP procesa correctamente el metadato y lo almacena en un directorio con el resto de metadatos de proveedores conocidos.
Declaración *Authentic* como IdP del SP implementado.	6322	OK	El SP procesa correctamente el metadato, lo almacena en un directorio y añade al IdP en su lista de proveedores de confianza.

Tabla 6.8: Pruebas I del SP con el IdP.

6.3.3. Pruebas de la interfaz gráfica de usuario e interoperabilidad

Estas pruebas se han realizado tras integrar las funciones de *IdM Library* en la aplicación CGI. En este caso, construimos un fichero de metadato propio para realizar la configuración del SP, en el cual indicamos las URLs de nuestro servidor para recibir los asertos enviados por el IdP y atender las peticiones de SLO. En este metadato se incluyen los perfiles de SSO y SLO y los "bindings" soportados por nuestro SP. Otro aspecto a destacar, es que se utiliza el mismo certificado para los propósitos de cifrado y firma.

Por otra parte, en esta sección también se presentan los resultados obtenidos en las pruebas realizadas para la interfaz de configuración del metadato del SP. En la tabla 6.9 se recogen las distintas pruebas llevadas a cabo durante esta fase.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Página de inicio de SSO	6331	OK	Se capturan correctamente los eventos de pulsar el botón de *Login* con los dos tipos de "bindings". Posteriormente se ejecutan las acciones correspondientes (inicio de SSO y establecimiento de la comunicación con el IdP).
Formulario con datos personales de la página de inicio de SSO	6332	OK	Se procesan correctamente los datos introducidos por el usuario y se almacenan en una estructura en la aplicación CGI.
Envío de mensaje de petición de autenticación al IdP tras pulsar botón de *Login*.	6333	OK	Al principio tuvimos algunos problemas para que el IdP verificara la firma XML, contenida en este mensaje, debido a que se producían problemas por la codificación de algunos caracteres del fichero de metadato como por ejemplo "_". Tras el eliminar este carácter del metadato del SP, comprobamos en la consola donde se encuentra lanzado el servidor del IdP (escuchando peticiones a través del puerto 3002), que éste verifica la firma y envía una respuesta con el *artifact* al SP.
Parseo de la *query string*.	6334	OK	Se identifica el tipo de acción a realizar en cada momento: que el usuario había pulsado un botón de login, que se debía recibir el aserto del IdP, iniciar el SLO, etc.
Prueba del perfil **SSO Web iniciado por el SP: POST-*Artifact* *Bindings***, usando "binding" `HTTP-Redirect` en el SP.	6335	OK	Esta prueba se ha realizado autenticando usuarios con y sin rol de administración en el IdP, a través de distintos navegadores. Necesario borrar *cookies* del navegador.
Prueba del perfil **SSO Web iniciado por el SP: POST-*Artifact* *Bindings***, usando "binding" `HTTP-POST` en el SP.	6336	OK	Esta prueba se ha realizado autenticando usuarios con y sin rol de administración en el IdP, a través de distintos navegadores. Necesario borrar *cookies* del navegador.

Interfaz de configuración de metadato para el SP	6337	OK	Se procesan adecuadamente los datos proporcionados por el usuario y se proporciona un enlace para visualizar el metadato generado. Dicho metadato se almacena en un fichero. En caso de que los datos introducidos por el usuario no sean correctos, se muestra la página de error correspondiente.

Tabla 6.9: Pruebas de la interfaz gráfica de usuario.

Una vez verificamos el correcto funcionamiento del proceso de *Single Sign On*, consideramos el siguiente escenario para probar la funcionalidad del perfil de *Single Logout* (SLO) utilizando el "binding" `SOAP`. En primer lugar se realizaron las pruebas de SLO con un único usuario autenticado en el sistema. A continuación se realizaron pruebas con varios usuarios autenticados simultáneamente, con el fin de comprobar si el SP es capaz de identificar correctamente qué usuario ha solicitado el cierre de sesión único.

En la tabla 6.10 se recogen los resultados de las pruebas realizadas para este escenario.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Un usuario ha completado el proceso de SSO y solicita un cierre de sesión único.	6338	OK	El SP obtiene de las *cookies* del navegador un identificador y encuentra en el sistema de ficheros de sesión la información relativa del usuario.
Varios usuarios se encuentran autenticados en el sistema y han solicitado cierre de sesión.	6339	OK	El SP identifica correctamente las solicitudes de estos usuarios, y se llevan a cabo los cierres de las sesiones. Cada usuario se ha autenticado a través de un tipo de navegador diferente.

Tabla 6.10: Pruebas II del SP con el IdP.

6.4. Otras pruebas de interoperabilidad

Además de las pruebas presentadas hasta ahora, realizamos un conjunto de pruebas de interoperabilidad teniendo en cuenta todas las entidades que forman parte nuestra plataforma de gestión de identidad (ver Figura 3.1). Como puede verse en el capítulo 3, hemos utilizado la notación SP_1, para referirnos al SP de ZXID, y SP_2, para el proveedor de servicios implementado. En esta fase, se realizó un conjunto de pruebas en el que un usuario de prueba solicita los servicios de SP_1 y SP_2 a través de SSO con distintos "bindings". Posteriormente, puede iniciar el cierre de sus sesiones activas desde SP_1 o desde SP_2.

En la tabla 6.11 se reflejan los resultados de las pruebas más relevantes realizadas durante esta fase.

Prueba realizada	**Código**	**Resultado**	**Observaciones**
Usuario solicita primero servicio de SP_1 y a continuación, de SP_2 y viceversa.	641	OK	Cuando el usuario solicita servicio de SP_1, es redirigido al IdP para introducir sus credenciales. Si se ha autenticado correctamente, accede al servicio solicitado. Cuando el usuario solicita el servicio de SP_2, obtiene el acceso directamente sin tener que volver a proporcionar sus credenciales al IdP, es decir, se ha realizado el inicio de sesión único. Se obtienen los mismos resultados si el usuario solicita primero un servicio con SP_2 y después con SP_1. Esta prueba se ha llevado a cabo empleando distintas combinaciones de los "bindings" soportados por los SPs para realizar la SSO. Es necesario limpiar las *cookies* del navegador si se quiere repetir esta prueba con otro usuario distinto.
Usuario solicita cierre de sesión único iniciado desde SP_1, con el botón *Local Logout*.	642	OK	Regresamos a la página de inicio de SP_1 y si nos colocamos en la ventana del servicio de SP_2, continúa la sesión.
Usuario solicita cierre de sesión único iniciado desde SP_2, con el botón *Single Logout (SOAP)*.	643	OK*	Regresamos a la página de inicio de SP_2, pero el servidor de SP_1, se queda esperando durante bastante tiempo. Puede ser algún problema de interoperabilidad de ZXID.

Tabla 6.11: Otras pruebas de interoperabilidad.

Capítulo 7

Historia del proyecto

Este proyecto surge a partir de un trabajo de investigación realizado dentro de GAST (Grupo de Aplicaciones y Servicios Telemáticos), en el Departamento de Telemática de la Universidad Carlos III de Madrid. Durante los meses de Noviembre de 2008 a Octubre de 2009 ha habido distintos períodos de desarrollo, cuya explicación es objeto de este apartado. En este capítulo, trataremos de explicar las diferentes etapas del proyecto, comentando los objetivos y el trabajo llevados a cabo en cada una de ellas, así como los distintos problemas encontrados en el camino y los replanteamientos y nuevos enfoques que se formularon para solventarlos.

Cabe destacar, que todas las pruebas documentadas en el capítulo 6, fueron realizadas en paralelo durante las etapas de despliegue y desarrollo de este proyecto.

7.1. Fases del proyecto

7.1.1. FASE I: Definición de requisitos.

- **Descripción de las tareas realizadas.**

  La proposición inicial sobre la que trabajar era poner en marcha una plataforma que nos permitiera probar la funcionalidad de un sistema de gestión de identidad. A partir de este planteamiento, comenzamos a definir requisitos:

  - Los aspectos de seguridad e identificación básicos debían estar garantizados de forma robusta, estos aspectos incluyen: autenticación,

autorización, integridad, confidencialidad y no repudio.

- Era necesario definir en la infraestructura de la plataforma un entorno de federación de identidades con mecanismos de *single sign-on*.
- Para permitir federación de identidad, la plataforma se basaría en el lenguaje de asertos y la infraestructura definida por OASIS, **SAML** .
- Una vez desplegados los distintos elementos del sistema se debía probar la interoperabilidad entre distintos proveedores.
- La gestión de usuarios debía realizarse de forma eficiente.

Partiendo de estos requisitos, se decidió que el modelo de seguridad se basaría en una infraestructura de PKI, con el objetivo de ofrecer un mayor nivel de seguridad en entornos no centralizados.

Tras esta formulación inicial de los requisitos fundamentales, comenzamos un proceso de documentación para adquirir un cierto conocimiento acerca del funcionamiento de las especificaciones de SAML. También se realizó un estudio de distintas implementaciones de código libre que existen de las dos iniciativas en gestión de identidad mencionadas. Para ello, tuvimos en cuenta aspectos de: funcionalidad implementada, versiones de los protocolos soportados, la integración con el sistema subyacente de gestión de certificados, la complejidad, la disponibilidad para diferentes distribuciones Linux, etc.

- **Resultados.**

  Como resultado de esta fase de documentación, decidimos seleccionar dos marcos de trabajo: **ZXID** y **Lasso**. ZXID, como hemos comentado en diversas ocasiones, proporciona el soporte necesario para los SPs y Lasso permite desarrollar tanto SPs como IdPs. La librería Lasso cuenta con un IdP denominado Authentic que está recomendado por ZXID para ser interoperable con su SP.

7.1.2. FASE II: Instalación y configuración de ZXID.

- **Descripción de las tareas realizadas.**

  En esta fase comenzamos con la instalación de la librería que nos proporcionaría el soporte para el SP. En este caso, fue necesario realizar de nuevo una labor de documentación acerca de los requisitos y dependencias externas para llevar a cabo la instalación de la misma.

Instalamos primero varias dependencias externas de ZXID: LibCurl, OpenSSL y Zlid. A continuación, dado que el SP ofrece sus servicios a través de un servidor Web, era necesaria la instalación de un servidor web capaz de soportar conexiones HTTPS. Aunque en la documentación de ZXID se recomendaba utilizar una versión simplificada del servidor Apache y más sencilla, denominada *mini-httpd*, decidimos elegir el servidor Apache, en su versión más reciente, porque podríamos reutilizar la instalación para crear distintas instancias de SPs e IdPs y está mejor documentado.

Realizamos un nuevo proceso de documentación centrado en los siguientes aspectos: configuración del servidor Apache con OpenSSL y LibCurl, así como su integración con ZXID.

Una vez que tuvimos todas las dependencias instaladas, pudimos proceder a instalar la librería que nos ofrecería el soporte del SP: ZXID. Durante este período, se generaron también diversos certificados digitales, necesarios para la configuración de Apache con soporte SSL y para la configuración de ZXID.

Después, realizamos la configuración del proveedor de servicios. También realizamos algunas pruebas prácticas con distintos ejemplos proporcionados por esta librería. Uno de los servicios ofrecidos incluye una interfaz gráfica sencilla que permite añadir la URL del metadato del IdP y realizar SSO. Sin embargo, ZXID no proporciona el soporte necesario para desarrollar IdPs, aunque recomienda un IdP llamado *Authentic* de Lasso.

- **Problemas encontrados.**

  Durante esta fase del proyecto tuvimos bastantes problemas para integrar la librería LibCurl con la versión 0.28 de ZXID. Sin embargo, al descargarnos la versión 0.29 pudimos compilar ZXID teniendo en cuenta esta dependencia externa sin problemas, por lo que debía tratarse de un *bug*.

  También tuvimos problemas para visualizar el metadato del SP, reflejados en la prueba *6111* (ver tabla 6.1), que finalmente, conseguimos solucionar.

- **Resultados.**

  Al finalizar esta estapa tenemos en funcionamiento el proveedor de servicios y nos hemos familiarizado con algunas de las funcionalidades de la librería ZXID. Sin embargo, al no contar con el soporte necesario para desarrollar un IdP, necesitamos instalar una nueva librería, así como sus dependencias externas para poder probar el mecanismo de SSO.

7.1.3. FASE III: Instalación y configuración del IdP Authentic.

- **Descripción de las tareas realizadas.**

  En este período instalamos el paquete que nos proporcionaría el soporte para el IdP: *Authentic*. De nuevo, fue necesario realizar una tarea de documentación acerca de los requisitos y dependencias externas para llevar a cabo la instalación de la misma. En este caso, el número de dependencias requeridas había aumentado, aunque no obstante, podíamos reutilizar la instalación de Apache con soporte SSL, realizada durante la fase anterior, para crear una nueva instancia.

  Sin embargo, tuvimos que añadir a la instalación y configuración del servidor Web nuevas dependencias para que trabajar con Python, debido a que parte del código del IdP está escrito en este lenguaje, y con SCGI, pues una vez instaladas todas las dependencias, cuando se lance el servidor éste se encontrará escuchando en el puerto 3002 las peticiones y respuestas de los SPs.

  Tras completar la instalación de *Authentic* y de sus dependencias, el siguiente paso consistía en realizar la configuración del IdP. Para esta tarea, fue necesario generar una clave pública y una clave privada con la ayuda de OpenSSL. También se configuraron opciones como el almacenamiento de usuarios, que en esta fase se definió que fuera a través de un sistema de ficheros.

- **Resultados.**

  En este momento, ya disponemos de dos elementos básicos en un sistema de gestión de identidad en funcionamiento: el SP y el IdP. Sin embargo, para probar funcionalidades y perfiles de SAML, como el mecanismo de *single-sign-on* o *single-logout*, es necesario el establecimiento de una relación de confianza entre los mismos, con el fin de que estos puedan interactuar.

7.1.4. FASE IV: Integración del SP y el IdP.

- **Descripción de las tareas realizadas.**

  Una vez teníamos el Proveedor de Servicios y el Proveedor de Identidad instalados y configurados, el siguiente paso, era integrarlos para probar los perfiles de SSO y SLO. Tambíen realizamos pruebas con diferentes servicios de ZXID. En la documentación encontramos que contaba con un WSP de ejemplo, lo cual resultó interesante para realizar pruebas.

- **Problemas encontrados.**

  En esta etapa del proyecto nos encontramos con más problemas de los esperados, pues pese a que en la documentación de ZXID figuraba que el IdP proporcionado por Lasso era interoperable con su SP, el establecimiento de una relación de confianza entre ambos, no fue un proceso sencillo.

  Encontramos problemas de incompatibilidad en la estructura de los metadatos, mencionados en los capítulos 4 y 6, y fue necesario realizar modificaciones tanto en el código fuente de ZXID como de *Authentic*. Para realizar estas modificaciones, fue necesario hacer un estudio exhaustivo de la funcionalidad de distintos métodos de estas dos librerías y hacer un seguimiento del curso de ejecución de los programas que implementan los roles del SP y el IdP, para poder localizar aquellos puntos del código en los que es preciso introducir cambios y saber exactamente qué cambios debíamos realizar. También resultó indispensable llevar a cabo procesos de documentación centrados en las especificaciones de SAML. Estos problemas se encuentran reflejados en los pruebas con código *6211*, *6212*, *6221* y *6222* (ver tablas 6.4 y 6.5). Por otro lado, cuando iniciamos las pruebas de los servicios web proporcionados por ZXID aunque la interfaz gráfica se podía cargar en el navegador, se producían errores al realizar la SSO. Tras realizar un estudio del código, encontramos que el problema se producía porque se requería un servidor de descubrimiento. Sin embargo, este servidor de descubrimiento no está disponible como código abierto, sino como un producto comercial de ZXID, por tanto, las pruebas de este servicio no se pudieron concluir con éxito. Estos problemas se encuentran reflejados en las pruebas *6114* y *6115* (ver tabla 6.1).

- **Resultados.**

  Superados los problemas a la hora de integrar los servicios existentes y desarrollar e implementar las funcionalidades de las que carecían, pudimos realizar pruebas de SSO y SLO con resultados satisfactorios.

  Sin embargo, el hecho de que fuera necesario un servidor de descubrimiento comercial para poder realizar puebas de SSO Web, nos llevó a descartar la idea de utilizar la librería ZXID para este propósito y surgió un nuevo planteamiento: implementar nuestro propio proveedor de servicios a partir de la experiencia adquirida hasta el momento y con ayuda de funciones proporcionadas por la librería Lasso. Esto nos permitiría realizar pruebas con el "binding" SOAP y estudiar más a fondo las especificaciones de SAML.

7.1.5. FASE V: Implementación del Proveedor de Servicios.

- **Descripción de las tareas realizadas.**

  El último de los problemas encontrado en la etapa anterior nos llevó a la opción de implementar nuestra propia librería de seguridad, basada en las especificaciones de SAML, de modo que introducimos un nuevo proveedor en nuestra plataforma y ésta pueda ser reutilizada e integrada en trabajos futuros. Además, diseñamos una interfaz gráfica de usuario sencilla, para poder realizar pruebas de SSO Web y SLO y de este modo comprobar la interoperabilidad del SP desarrollado con el IdP configurado en la FASE III. También podremos realizar pruebas de interoperabilidad con el SP configurado en la FASE II.

  En esta fase, fue necesario realizar un nuevo proceso de documentación centrado en los siguientes aspectos: el estudio de las tareas que debe realizar un SP y del API de la librería Lasso, poniendo especial atención en aquellas funciones que nos permitirían la construcción, el envío y la recepción de mensajes para comunicarnos con el IdP, así de como las estructuras y funciones que nos facilitarían la implementación de los perfiles de SSO y SLO.

  En esta etapa de documentación, encontramos un SP de ejemplo proporcionado por la librería Lasso, que contribuyó en gran medida a clarificar el funcionamiento y la utilización de las estructuras y funciones relacionadas con los perfiles de SSO y SLO en Lasso. Tras este estudio también llegamos a la conclusión de que la parte de transporte de los mensajes de peticiones y respuestas de autenticación a través de HTTP/HTTPS no estaba implementada en la librería Lasso.

  Esto nos llevó a realizar un estudio de librerías en C que implementen los protocolos HTTP y HTTPS. Se decidió utilizar la librería *LibNeon* para este fin.

  Después de obtener el conocimiento necesario para abordar el desarrollo de nuestro SP, comenzamos con su implementación.

- **Problemas encontrados.**

  En esta fase no encontramos ningún problema relevante que impidiese continuar con el desarrollo del SP según lo previsto. Sin embargo, observamos que el método `ne_read_response_block()` de la librería LibNeon, tenía una condición errónea que provocaba la terminación prematura de la lectu-

ra de las respuestas HTTP/HTTPS, aspecto que reflejamos también en el capítulo 6, en la prueba *6316* (ver tabla 6.7).

- **Resultados.**

  El resultado de esta etapa es una librería que soporta los casos de SSO Web y SLO entre proveedores utilizando pseudónimos de tipo transitorio. De acuerdo con esto, cualquier SP que utilice el API desarrollado no tendrá que mantener una base de datos de cuentas usuarios, ya que la gestión de las mismas es llevada a cabo exclusivamente por el IdP.

7.1.6. FASE VI: Instalación y configuración del servidor LDAP. Integración con el IdP.

- **Descripción de las tareas realizadas.**

  Hasta este momento, tenemos en funcionamiento dos de los elementos básicos en un sistema de gestión de identidad: el SP y el IdP. Además, hemos introducido un nuevo SP y para ello, hemos desarrollado nuestra propia librería. Sin embargo, si recordamos, uno de los requisitos del proyecto era: "realizar una gestión eficiente de los usuarios que participen en el sistema". En el inicio de esta etapa, la información de identidad de los usuarios se encontraba almacenada en un sistema de ficheros. Esta opción no resulta eficiente si el número de usuarios que debemos considerar es elevado.

  Si analizamos nuestra plataforma, encontramos que la entidad con mayor carga en las tareas gestión de usuarios es el IdP, pues por ejemplo, como hemos comentado anteriormente, los SPs que utilicen el API desarrollado en la fase anterior no tendrían que preocuparse de la gestión de cuentas de sus usuarios si no lo desean.

  Por lo que se debe proporcionar al IdP un servicio de directorio LDAP para gestionar los usuarios, lo que permitirá mejorar el rendimiento y la escalabilidad de nuestro sistema.

  Este planteamiento requiere un nuevo proceso de documentación centrado en varios aspectos: el análisis de las especificaciones del protocolo LDAP, el estudio de herramientas de código libre que nos puedan proporcionar esta funcionalidad y cómo podíamos realizar la integración con nuestro IdP.

  En la documentación de *Authentic* (el soporte del IdP), encontramos que este paquete nos proporciona una interfaz gráfica para realizar la configuración de los parámetros de nuestro servidor LDAP y se recomienda utilizar la herramienta de código abierto OpenLDAP.

Con la información y el conocimiento obtenidos tras el proceso de documentación, realizamos la instalación y configuración de un servicio de directorio LDAP, basado en OpenLDAP. A continuación integramos en el IdP este nuevo servicio.

Además, a fin de facilitar las tareas de inserción, búsqueda, borrado, etc, de usuarios en el servidor, instalamos y configuramos un explorador de LDAP, basado en la herramienta de código libre *phpLDAPadmin*. También realizamos pruebas de introducción de nuevos usuarios mediante esta nueva opción.

- **Problemas encontrados.**

  En esta fase tampoco surgieron problemas relevantes que nos impidieran proporcionar al IdP el servicio de directorio LDAP según lo previsto. Sin embargo, encontramos algunos problemas a la hora de realizar la integración entre ambos. En concreto, que la inserción de nuevos usuarios no se podía realizar directamente desde el IdP. Éste era capaz de realizar una lectura correcta de los datos almacenados en el directorio LDAP, pero no podía llevar a cabo las operaciones de escritura en el mismo.

- **Resultados.**

  Al finalizar de esta etapa, conseguimos que el IdP dispusiera de un servicio de directorio LDAP y de este modo, poder realizar una gestión más eficaz de los usuarios de nuestro sistema.

  Respecto a la inserción de usuarios, se optó por realizarla directamente desde el explorador de directorios LDAP.

7.1.7. FASE VII: Documentación.

Esta fase consiste en la documentación de todo el trabajo realizado, acción materializada en la presente memoria.

7.2. Resumen

Como complemento a la información proporcionada en las secciones anteriores, y con el fin de mostrar el tiempo invertido en cada una de las fases descritas, presentamos la tabla 7.1. Hay que mencionar que el trabajo se realizó durante los meses de Noviembre de 2008 a Octubre de 2009.

Como se puede apreciar en la tabla 7.1, los procesos de instalación, configuración e integración del SP y el IdP supusieron una gran inversión de tiempo, debido a que conseguir la interoperabilidad entre los mismos supuso el estudio y modificación de código fuente a partir de fallos en los que se proporcionaba muy poca información. Además, la documentación disponible de la librería ZXID era relativamente escasa y el código fuente está programado de manera que resulta bastante complicado seguir el proceso de ejecución de los distintos métodos. En cambio, la experiencia con la librería Lasso durante estas fases fue más positiva: el código estaba mejor estructurado y resultaba fácil de entender y además, en diversas ocasiones comprobamos que el soporte ofrecido por esta librería con usuarios no de pago, era bastante aceptable.

Fase	**Descripción**	**Duración**
Fase I	Definición de requisitos	2 semanas
Fase II	Instalación y configuración de ZXID	1 mes
Fase III	Instalación y configuración del IdP Authentic.	1 mes
Fase IV	Integración del SP y el IdP	3 meses
Fase V	Implementación del Proveedor de Servicios	3 meses
Fase VI	Instalación y configuración del servidor LDAP. Integración con el IdP	2 semanas
Fase VII	Documentación	2 meses

Tabla 7.1: Duración asociada a cada fase del proyecto

Utilizamos Lasso como una de las bases para desarrollar nuestra propia librería y poder implementar nuestro propio SP. Tras estar familiarizados con el entorno de trabajo, las tareas de desarrollo resultaron más sencillas, aunque al tratarse de un nuevo planteamiento, supuso un nuevo esfuerzo de documentación. Por esta razón, algunos aspectos de las funciones de la librería y de la interfaz gráfica de usuario han sido programados de la forma más sencilla, de manera que existen varias posibilidades de ampliación y mejora. De estas posibles líneas de trabajo futuro dejamos constancia en 8.2.

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

La gestión de identidad ha surgido en los últimos años como un concepto clave para aliviar a las empresas de esta tarea y permitirles centrarse en su núcleo de negocio. Actualmente, existen diversas iniciativas para establecer un marco de trabajo que ofrezca, a los desarrolladores de servicios, mecanismos básicos de seguridad e identidad. Además estas iniciativas no se han elaborado de espaldas al usuario final, y las infraestructuras también tienen en cuenta la flexibilidad y la facilidad de uso. El ejemplo más notable es el mecanismo de *Single Sign-On*, que permite al usuario autenticarse frente a un único proveedor de identidad que es el encargado de mediar con los sitios federados para comunicar la identidad del usuario sin necesidad de requerir nuevas autenticaciones del usuario.

En este proyecto, hemos hecho un estudio del estándar SAMLv2 y para ello, hemos desplegado una plataforma de gestión de identidad federada que nos ha permitido realizar pruebas en una primera instancia de los casos de uso de SSO y SLO con distintos *bindings* definidos en esta especificación. Con el fin de proporcionar un entorno de pruebas adecuado para el presente proyecto seleccionamos dos "frameworks": ZXID y Lasso. ZXID para proporcionar el soporte necesario para los SPs y Lasso para desarrollar tanto SPs como IdPs.

Sin embargo, en un escenario de gestión de identidad, para que se pueda llevar a cabo la comunicación entre SP e IdP es necesario que exista una relación de confianza entre ellos. Por lo que ha sido necesario configurar de manera estática cada entidad para que reconozca qué entidades son fiables. De este modo, los distintos proveedores no pueden ofrecer sus servicios de forma inmediata, ya que no pueden entablar relaciones sin pasar por un proceso de configuración.

Tanto los SPs como el IdP se ofrecen a través de un servidor Web. Para realizar el despliegue de estos proveedores, fue necesario instalar y configurar diversas dependencias externas, como puede verse en el apéndice B. Una vez tuvimos el Proveedor de Servicios y el Proveedor de Identidad instalados y configurados, el siguiente paso, el de integrar y hacer *Single Sign On* (SSO) trajo más problemas de lo esperado, como se comentó en los capítulos 6 y 7. Se dieron, por ejemplo, problemas de incompatibilidad en la estructura de los metadatos y se tuvieron que realizar cambios en el código fuente de ZXID y Lasso. Para ello, fue necesario trabajar con varios lenguajes de programación y distintos "frameworks" (C, *Python*, *Openssl*, *Libxml*, *Quixote*, etc). Superados estos problemas a la hora de integrar los servicios existentes y desarrollar e implementar las funcionalidades que carecían, conseguimos completar con éxito las pruebas de los perfiles de SSO y SLO.

Además, desarrollamos nuestro propio proveedor de servicios y pudimos realizar pruebas de interoperabilidad entre los proveedores de nuestra plataforma, utilizando como clientes distintos navegadores web con resultado exitoso. Por lo que podemos concluir, que los exploradores utilizados (*Firefox*, *Internet Explorer*, *Konqueror*, etc.) soportan las cabeceras de las tramas HTTP/HTTPS intercambiadas durante el diálogo SAML llevado a cabo entre un SP y un IdP.

En lo que se refiere a la cobertura del estándar, nuestro sistema soporta las dos variantes del perfil de SSO Web iniciado por el SP, contempladas en las especificaciones de SAML. En el perfil SSO: POST/*Artifact Bindings*, el SP típicamente utiliza el "binding" `HTTP-Redirect` para enviar una solicitud de autenticación al IdP, que es como se implementa en ZXID. El proveedor de servicios que hemos desarrollado, nos permite también, mandar esta petición usando el "binding" `HTTP-POST`, lo cual se recomienda en la especificación para aquellos casos en los que el tamaño del mensaje sea considerable (por ejemplo, contiene muchos campos opcionales). Además, nuestra plataforma soporta el perfil de SLO y permite realizar el caso de uso de cierre de sesión único iniciado desde el SP con múltiples SPs. También ofrece cobertura para el caso de uso de federación utilizando identificadores de tipo transitorio.

Finalmente, para cumplir con el objetivo de realizar una gestión de usuarios eficiente, hemos dotado al IdP de un servicio de directorio LDAP.

8.2. Líneas futuras

En esta sección proponemos una serie de mejoras y trabajos futuros que se podrían realizar en la plataforma de gestión de identidad desplegada. A conti-

nuación presentamos algunas líneas de trabajo futuro:

- **Soporte de otros casos de gestión de identidad federada.**

  En SAML se contemplan otros casos de uso, como son los caso de **federación utilizando identificadores de tipo permanente** y **defederación**. En nuestro escenario, la única entidad que dispone de un servidor LDAP es el proveedor de identidad, debido a que comentamos, es el que mayor carga soporta en las tareas de gestión de usuarios. Sin embargo, un SP puede mantener tanta información de usuarios como requiera de acuerdo a los servicios ofertados y al caso de federación utilizado. En el caso de federación abordado en este proyecto, los SPs permiten proporcionar servicios anónimos de carácter *one-time*, por lo que no es necesario que mantengan información de cuentas de usuarios, sino que simplemente almacenan los identificadores creados durante el transcurso de cada sesión. Pero si se permitiera la creación automática de cuentas, entonces sí que habría que gestionar datos del usuario en los SPs, para lo cual resultaría muy ventajoso que estos disponga de un servicio LDAP, a fin de mejorar la eficiencia y el rendimiento del sistema.

  Además, podríamos extender el API de la librería desarrollada para el SP y añadir soporte para estos casos de uso.

- **Ampliar la cobertura de perfiles.**

  - Resultaría interesante realizar pruebas acerca del funcionamiento del perfil SSO Web iniciado desde el IdP usando el "binding" `POST`. Se podrían estudiar las diferencias existentes, en cuanto al contenido de los mensajes intercambiados entre los proveedores, su interoperabilidad, etc, respecto al caso de iniciar el proceso de SSO desde el SP.
  - En las especificaciones de SAML también se define un perfil, denominado *Enhanced Client and Proxy* (ECP), que permite realizar SSO sin que sea necesario el uso de un navegador web. Por lo que se podría realizar un proceso de documentación acerca del soporte que proporciona la librería Lasso para este perfil, con el fin de extender el API de la librería desarrollada. Además, habría que estudiar el soporte de *Authentic* para este caso, de modo que podamos saber si se pueden hacer pruebas de interoperabilidad con SPs y este IdP.

- **Proporcionar al IdP soporte *proxy*.**

  En el capítulo 4 comentamos que el proveedor de identidad *Authentic* tiene la posibilidad de actuar como servidor *proxy*. Esta opción puede resultar

interesante si realizamos pruebas de interoperabilidad con otros IdPs, de modo, que nuestro proveedor de identidad pueda actuar como un servidor *proxy*. De este modo, podríamos realizar pruebas en escenarios más complejos.

- **Extensión del Proveedor de Servicios.**

  El diseño actual presenta una interfaz web muy sencilla que permite al usuario realizar SSO Web y SLO, y le notifica el resultado de estas operaciones. Sin embargo, el proceso de configuración del SP se realiza a partir de la lectura de certificados y metadatos que previamente deben estar situados en un sistema de ficheros. Para facilitar el uso de esta librería, sería deseable:

  - Obtener las rutas de los ficheros requeridos de un archivo de configuración rellenado por el usuario y extender la interfaz de configuración web. En dicha interfaz, se podrían presentar diferentes menús que permitieran al usuario:
    - Proporcionar el metadato, certificado y cadena de certificados del IdP.
    - Un ejemplo del formato de fichero de configuración que debe rellenar.
  - Obtener los certificados a partir de los metadatos, puesto que para configurar las entidades de nuestro sistema, actualmente es necesario proporcionar los ficheros de metadato y certificado de las mismas. Sin embargo, dado que el certificado es parte de la información que contienen los metadatos, sería interesante crear nuevas funciones que extraigan este material criptográfico de los metadatos y reducir el número de datos de entrada necesarios en los distintos proveedores.

- **Pruebas.**

  Podrían realizarse pruebas más exhaustivas de cara a garantizar la robustez de la librería desarrollada para proporcionar el soporte del SP y la interoperabilidad del mismo con otros proveedores. Resultaría interesante realizar este tipo de pruebas con un IdP creado por Google basado en SAML. El lector puede encontrar más información acerca de este IdP en [53], si lo desea.

Apéndice A

Presupuesto

En este capítulo presentamos el presupuesto asociado al desarrollo del presente proyecto. En primer lugar, se detallan los costes de personal en que se ha incurrido y, a continuación, se muestran los costes derivados del material empleado. La adición de cada uno de estos costes constituirá el presupuesto final.

A.1. Costes de personal

Los costes de personal incluyen los honorarios del Ingeniero de Telecomunicación encargado del desarrollo del proyecto. Aunque el ritmo de trabajo no ha sido constante, se estima que se han invertido aproximadamente 5 horas al día durante los meses de Noviembre de 2008 a Octubre de 2009. Hemos considerado que el número de días laborables en un mes es igual a 20, lo que hace un total de 1100 horas.

Según el baremo orientativo del COIT[54], los honorarios de un Ingeniero de Telecomunicación eran de 72 euros la hora en el año 2008. Sin embargo, debido a que este dato ya no es de carácter público, para tener en cuenta los honorarios en el año 2009, hemos aplicado un aumento del valor medio anual del IPC en dicho año, que como se puede consultar en [55] es del 0.6 %. Por tanto, teniendo en cuenta también el coeficiente de reducción del 40 % impuesto por superar 1080 horas, el coste total asciende a 52.099 euros. La tabla A.1 recoge este resultado.

Concepto	Horas	Honorarios	Importe
Ingeniero de Telecomunicación	1.100 horas	47 €/hora	52.099€

Tabla A.1: Costes de personal

A.2. Costes de material

Los materiales empleados durante la realización del proyecto han sido los siguientes:

- Un ordenador personal de sobremesa con sistema operativo Linux, valorado aproximadamente en 850 euros.
- El *software* utilizado es en su totalidad de libre distribución.
- Conexión a Internet durante la realización del proyecto. Ha sido necesaria para conseguir documentación y para realizar pruebas del sistema de gestión de identidad, ya que los distintos proveedores que intervienen en él ofrecen sus servicios a través de servidores web. Está valorada aproximadamente en 42 euros al mes, que multiplicado por los meses de trabajo, supone un coste de 420 euros.

En la tabla A.2 se detallan los costes de material.

Concepto	**Unidades**	**Precio unitario**	**Importe**
Ordenador personal	1	850€	850€
Conexión a Internet	1	420€	420€
TOTAL			**1.270€**

Tabla A.2: Costes de material

A.3. Presupuesto total

El presupuesto final para la realización de este proyecto está formado por los costes de material y de personal presentados anteriormente. Como se observa en la tabla A.3, el total asciende a 53.369 euros.

Concepto	**Importe**
Costes de personal	52.099€
Costes de material	1.270€
TOTAL	53.369€

Tabla A.3: Coste total

Apéndice B

Manual de instalación

Para la realización de este proyecto ha sido necesario instalar una serie de paquetes y librerías de código abierto, comentados durante los capítulos 3 y 4, tanto para obtener el soporte necesario para el proveedor de servicios y el proveedor de identidad, como para proporcionar un servicio de directorio LDAP. En esta sección se describen los pasos necesarios para llevar a cabo la instalación de la librería ZXID, el módulo Authentic, así como sus dependencias externas, y los paquetes de código abierto OpenLDAP y phpLDAPadmin.

B.1. ZXID

En este apartado se detallan los pasos que se deben seguir para instalar la librería ZXID. Esta librería tiene algunas dependencias externas, comentadas en 4 que de nuevo volvemos a mencionar:

- Apache con soporte SSL (versiones 1.3 o 2.x)
- Zlib.
- Openssl (versión 0.9.8 o superior).
- Libcurl (versión 7.15.X.X o superior).

Por lo tanto, a lo largo de las siguientes secciones se realizará una explicación acerca de cómo instalar estas dependencias y finalmente, se describirán los pasos necesarios para instalar ZXID.

B.1.1. Instalación de dependencias externas

B.1.1.1. Instalación del servidor Web Apache con soporte SSL

La instalación del servidor Apache con soporte de SSL se puede hacer en tres pasos si se tiene permisos de administrador:

- Instalación de Apache 2.2: `#apt-get install apache2`
- Instalación del módulo SSL para apache2: `#apt-get install libapache2-mod-ssl`
- Carga del módulo SSL: `#a2enmod ssl`

Si no se tienen permisos de administrador, entonces será necesario configurar, compilar e instalar los ficheros fuentes de la siguiente manera:

```
# tar xvzf httpd-2.2.11.tar.gz
# cd httpd-2.2.11
# ./config --prefix=$(RUTA_INSTALACION_APACHE) \
   --enable-ssl -with-ssl= $(RUTA_OPENSSL, suele ser /usr/local/ssl)
# make
# make test
# make install
```

B.1.1.2. Instalación de Zlib, LibCurl y OpenSSL

Para llevar a cabo la instalación del resto de dependencias de ZXID, explicamos los pasos a seguir tanto para un usuario administrador como para un usuario normal. Un usuario con permisos de administrador debe realizar los siguientes pasos con la ayuda del paquete apt-get:

- Instalación de zlib: `# apt-get install zlib`
- Instalación de openssl: `# apt-get install openssl`
- Instalación de libcurl: `# apt-get install libcurl`

Un usuario sin permisos de administrador necesitará instalar las librerías necesarias en espacio de usuario de la siguiente manera:

- Instalación de zlib:

```
# cd  $(RUTA_ZLIB)
# ./configure --prefix = $(RUTA_INSTALACION_LIBRERÍA)
# make
# make install
```

- Instalación de openssl:

```
# cd  $(RUTA_OPENSSL)
# ./configure --prefix = $(RUTA_INSTALACION_LIBRERÍA)
# make
# make install
```

- Instalación de libcurl:

```
# cd  $(RUTA_LIBCURL)
# ./configure --prefix = $(RUTA_INSTALACION_LIBRERÍA)
# make
# make install
```

B.1.2. Instalación de ZXID

Para instalar esta librería, se ha seguido la opción de instalar las dependencias externas con permisos de administrador para no tener que modificar el archivo Makefile de ZXID, ya que tiene las rutas por defecto de las librerías externas.

En esta sección se explica el modo de compilar e instalar la librería ZXID una vez que ya tenemos instaladas y configuradas correctamente todas sus dependencias externas. Se detalla cómo instalar la parte obligatoria; puesto que aunque también dispone de módulos opcionales que sirven para trabajar con Java, PHP y Perl, como en la realización del proyecto se iba a trabajar con C/C++ no eran necesarios. Se deben seguir los siguientes pasos:

1. Se compilan todos los ficheros y se crea una jerarquía de directorios que se describe a continuación:

   ```
   make
   make dir
   ```

2. Con la directiva make dir se crea la siguiente jerarquía de directorios:

   - En `/home/apt/usuario` se crea una carpeta llamada zxid que almacenará diversos ficheros y directorios

- `/home/apt/usuario/zxid/zxid.conf`: Fichero principal de configuración.
- `/home/apt/usuario/zxid/zxid.pem`: Certificado firmado que venía por defecto en el paquete ZXID. Este fichero contiene el certificado firmado y la clave privada en un único archivo.
- `/home/apt/usuario/zxid/pem/`: En este directorio se almacenan los certificados de nuestro Proveedor de Servicios (SP). Se encuentran los siguientes certificados, que venían por defecto en el paquete ZXID:

  ```
  ca.pem.
  enc-nopw-cert.pem.
  sing-nopw-cert.pem.
  logenc-nopw-cert.pem
  logsign-nopw-cert.pem.
  ```

  Se ha modificado el fichero zxid.pem, que es el que se utiliza para crear los certificados mencionados arriba, por un certificado propio. Este certificado es el mismo que se utiliza para configurar el servidor de Apache con SSL.
- `/home/apt/usuario/zxid/cot/`: Metadatos de los *partners* CoT (caché de metadatos). En este directorio se almacenan los ficheros con los metadatos de los Proveedores de Identidad (IdP) con los que el SP va a trabajar. En este directorio se encuentran una serie de metadatos de los partners CoT que venían por defecto en el paquete ZXID.
- `/home/apt/usuario/zxid/ses/`: En este directorio se almacenan los datos correspondientes a las sesiones de los usuarios
- `/home/apt/usuario/zxid/log/`: En este directorio se almacenan ficheros *Log*, *pid*, etc. Además hay dos subdirectorios llamados *issue* y *rely*.

3. Finalmente, se procede a la instalación ZXID y el módulo de autenticación basado en SAML para Apache mediante las siguientes instrucciones:

```
make install
make apachezxid
make apachezxid_install
```

B.2. Authentic

En este apartado se detallan los pasos que se deben seguir para instalar los paquetes o librerías requeridos, antes de proceder a instalar el módulo Authentic, que recordemos, es el que nos proporciona el soporte necesario del IdP. Este paquete tiene algunas dependencias externas, comentadas en el capítulo 4, que de nuevo volvemos a mencionar:

- Apache con soporte SSL (versiones 1.3 or 2.x, se recomienda Apache 2)
- Lasso (versión 0.6.3 o mayor)
- Quixote(versión 2.0 o mayor)
- Mod_python o SCGI (versión de SCGI 1.7 o mayor). Se recomienda utilizar el módulo SCGI.

Por lo tanto, a lo largo de las siguientes secciones se realizará una explicación de cómo instalar estas dependencias, a excepción de la primera, puesto que se ha comentado en B.1.1.1 y finalmente, se describirán los pasos necesarios para instalar el paquete Authentic.

B.2.1. Instalación de dependencias externas

Instalación de los módulos Python y SCGI

Antes de instalar y compilar el resto de las dependencias de Authentic, es necesario instalar una serie de paquetes de Python y SCGI para que éstas se puedan compilar e instalar correctamente. Debido a que algunas de estas dependencias deben integrarse con Apache, es conveniente que la instalación se realice como administrador, puesto que si no, sería necesario realizar una reinstalación completa de Apache para habilitar las dependencias de Python y SCGI.

- `sudo apt-get install python2.5`
- `sudo apt-get install pyhton2.5-dev` Este paquete contiene algunos ficheros que no contiene el paquete anterior, como por ejemplo Python.h y sin el cual, surgían problemas al compilar el paquete Quixote porque no encontraba ciertos tipos de datos y estructuras que están definidos en dicho fichero.
- `sudo apt-get install libapache2-mod-python` Con este paquete indicamos a Apache que trabaje también con el módulo Python.

- `sudo apt-get install libapache2-mod-scgi` Con este paquete indicamos a Apache que trabaje también con el módulo scgi.
- `sudo apt-get install python-scgi` Con este paquete integramos los dos módulos anteriores.

Instalación de Lasso La siguiente dependencia de Authentic es una librería denominada Lasso, escrita en C, de software libre y cuyo objetivo es aplicar las normas del estándar SAML y protocolos relacionados. Para la implementación de Lasso se han utilizado como librerías criptográficas libxml2, XMLSec y OpenSSL.

La versión más reciente de Lasso es la 2.2.2, sin embargo ésta todavía no es muy estable, pues al tratar de compilarla faltaban algunos archivos necesarios para la correcta instalación. Por lo que se ha instalado la versión anterior (2.2.1), la cual se puede descargar de la siguiente dirección:

`http://lasso.entrouvert.org/download`

Por temas propios de configuración de la máquina y esquema de montaje NFS, se va a realizar la compilación e instalación del paquete desde el directorio temporal.

La secuencia de comandos que se debe seguir es la siguiente:

```
# cd /tmp
# ./configure --enable-debugging
# make
# sudo make install
```

Cabe resaltar que al ejecutar el comando `./configure` se realiza un chequeo de las dependencias de Lasso y al principio surgieron problemas con las siguientes dependencias:

```
glib-2.0
gobject-2.0
libxml-2.0
xmlsecl
xmlsecl-openssl
```

En este procedimiento se instalaron las siguientes versiones de dichos paquetes:

```
sudo apt-get install libxmlsec-dev
sudo apt-get install libxml2-dev
sudo apt-get install libglib2.0-dev
sudo apt-get install swig1.3
sudo apt-get install libsasl2-2
sudo apt-get install libsasl2-dev
```

Instalación de Quixote Quixote, como se comentó en la sección 2.5.1.1, es una aplicación web para programadores que trabajen con Python y necesiten desarrollar sitios web dinámicos. Esta aplicación requiere la versión Python 2.0 o superior y un servidor web que implemente el protocolo CGI. La URL desde la que se puede descargar este paquete es la siguiente: `http://quixote.ca/download.html`. Nos hemos descargado la versión más reciente, que es la 2.6. Para instalar y compilar este nuevo paquete, se necesitan tener permisos de administrador, ya que por defecto se instala en `/usr/lib/python2.5` y es un poco complejo modificar los *scripts* para que este paquete se instale en otro directorio. Nos situamos en el directorio donde tengamos el código fuente e introducimos en línea de comandos las siguientes instrucciones:

```
# cd ($QUIXOTE) // Ruta donde se encuentra el código fuente.

# sudo python setup.py install
                              // setup.py lo compila e instala en
              // /usr/lib/python2.5/site-packages/quixote.
```

Para comprobar que la instalación de este paquete se ha realizado correctamente, se puede ejecutar el servidor de la siguiente manera:

```
# cd ($QUIXOTE)/server // Ruta donde se encuentra el código
                       // fuente del servidor de Quixote.
# ./simple_server.py //Lanzamos el servidor.
```

Si ponemos en un navegador la siguiente dirección: `http://localhost:8080` nos aparece un Hello World! y una demo de Quixote, por lo que comprobamos que el servidor funciona de modo correcto.

B.2.2. Instalación de Authentic

En este apartado se detalla el procedimiento a seguir para de compilar e instalar el paquete Authentic, que recordemos es el que implementa el rol del Proveedor de Identidad.

Se puede descargar la versión 0.7.1 de la siguiente URL: `http://authentic.labs.libre-entreprise.org/`. Para compilar e instalar este paquete, es necesario tener permisos de administrador, ya que por defecto se instala en /usr/lib/python2.5 y es un poco complicado modificar los *scripts* para que este paquete se instale en otro directorio. Nos debemos situar en el directorio donde tengamos el código fuente e introducir en línea de comandos las siguientes instrucciones:

```
# cd ($ AUTHENTIC) // Ruta donde se encuentra el codigo fuente.

# sudo python setup.py install
                 // compilacion e instalacion del paquete en
                // /usr/lib/python2.5/site-packages/authentic
```

Para finalizar, destacar que en /var/lib/authentic se irá creando una jerarquía de directorios, en los cuales, se almanacenan información relativa a los usuarios creados (login y password), a las sesiones, ficheros con el certificado y la clave privada del IdP, etc.

B.3. OpenLDAP

Para obtener el soporte necesario para proporcionar un servidor LDAP, de tal modo, que las tareas de gestión y almacenamiento de usuarios en el IdP sean más eficientes se seleccionaron los marcos de trabajo: OpenLDAP [40] y phpLDAPadmin [41], por los motivos expuestos en el capítulo 3.

Recordemos que OpenLDAP es una implementación de código libre que proporciona el soporte necesario para desarrollar clientes y servidores LDAP. Por otra parte, con el fin de facilitar las tareas de administración del servicio de directorio LDAP, se seleccionó el "framework" phpLDAPadmin, que recordemos se trata de una herramienta para la administración de servidores LDAP escrito en PHP, basado en interfaz Web.

En esta sección se describen los pasos necesarios para realizar la instalación y configuración de un servidor LDAP y de una interfaz Web que facilite las tareas de inserción, borrado y búsqueda de usuarios utilizando OpenLDAP y phpLDAPadmin.

B.3.1. Compilación e instalación de OpenLDAP

En este apartado se detalla el proceso que se debe seguir para realizar la compilación e instalación del paquete OpenLDAP a partir de los ficheros fuentes. Cabe resaltar que se ha de realizar este procedimiento a partir de los archivos fuentes y no con la ayuda de la herramienta *apt-get*, como en otras ocasiones, para proporcionar a LDAP soporte seguro a partir de la librería criptográfica OpenSSL.

Lo primero que debemos hacer, es descargarnos la última versión más estable de la página de descargas de OpenLDAP, disponible en la siguiente URL: `http://www.openldap.org/software/download/`. En el momento de desarrollo de este proyecto, dicha versión era la 2.4.16.

Una vez que tenemos el código fuente, como éste se encuentra comprimido en un archivo del tipo *tar.gz*, debemos elegir un directorio para descomprimirlo, por ejemplo $HOME/ldap, aunque cualquier ubicación de este directorio podría ser válida para este propósito. Debemos introducir la siguiente instrucción en la línea de comandos:

```
tar xvfz openldap-2.4.16.tgz
```

El siguiente paso que debemos realizar, es indicar las opciones de compilación e instalación que deseamos. En nuestro caso, mediante la siguiente instrucción:

```
./configure --with-tls --enable-debug --enable-cache --enable
-referrals --with-threads --enable-slapd
```

Con la opción `--with-tls` podremos tener soporte seguro a través de OpenSSL. La opción `--enable-debug` nos permitirá realizar un seguimiento más exhaustivo del proceso de compilación e instalación y nos proporcionará un mayor nivel de información en el caso de que se produzca algún tipo de error durante este proceso. Otra opción interesante, es `--enable-slapd`, que nos creará el ejecutable para el servidor OpenLDAP. También tenemos la posibilidad de indicar la ubicación del directorio en el que deseamos que se realice la instalación con la opción `--prefix`. Si no indicamos expresamente esta ubicación, por defecto los archivos binarios se instalarán en el directorio `/usr/local/libexec` y los ficheros de configuración en `/usr/local/etc/openldap`, por lo que deberemos cerciorarnos de que disponemos de permisos de administrador.

Las siguientes instrucciones que debemos introducir en la línea de comandos son:

```
make depend
make
```

```
cd tests
make tests
cd ..
sudo make install
```

De nuevo, comentamos que utilizamos la directiva `sudo`, porque los ficheros binarios y de configuración quedarán ubicados en los directorios por defecto, ya que no hemos especificado un directorio de instalación al realizar el `configure`. Finalizado el proceso de instalación del paquete, el siguiente paso, es crear un archivo que contemple la información de configuración de nuestro servidor LDAP. Sin embargo, debido a que la creación de este fichero requiere cierta complejidad, dedicaremos la siguiente sección a explicar detalladamente este proceso.

B.3.2. Construcción del fichero de configuración

Para poner en funcionamiento nuestro servidor LDAP, es necesario definir un fichero de configuración llamado *slapd.conf*, que se encontrará situado en `/usr/local/etc/openldap/`. Este archivo está estructurado en dos partes, una primera con las opciones globales del OpenLDAP y otra con las definiciones de cada base de datos (directorio) que deseemos tener. Cabe destacar que las opciones de este archivo deben comenzar en la primera columna del fichero, ya que si no lo hacen así, serán considerados continuaciones de la linea anterior.

A continuación se muestra un ejemplo del fichero *slapd.conf* para definir un directorio que contendrá una base de datos de usuario para un sistema de autenticación:

slapd.conf

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include     /usr/local/etc/openldap/schema/core.schema
include     /usr/local/etc/openldap/schema/cosine.schema
include     /usr/local/etc/openldap/schema/inetorgperson.schema
include     /usr/local/etc/openldap/schema/nis.schema
include   /usr/local/etc/openldap/schema/userauthentic.schema

# Define global ACLs to disable default read access.
# Do not enable referrals until AFTER you have a working directory
# service AND an understanding of referrals.
#referral   ldap://root.openldap.org
pidfile     /usr/local/var/run/slapd.pid
argsfile    /usr/local/var/run/slapd.args
```

slapd.conf

```
#######################################################################
# BDB database definitions
#######################################################################
database    bdb

# Base de datos por defecto a usar ldbm
#database ldbm

# Aqui es donde se guardara los datos del directorio
directory   /usr/local/var/openldap-data

# Sufijo o raiz(root) del directorio. Es el nodo raiz superior
# de nuestro directorio ldap

suffix      "dc=gast.it.uc3m,dc=es"

# El nombre distinguido del directorio manager
rootdn      "cn=admin,dc=gast.it.uc3m,dc=es"

# replication directives
 replica uri=ldap://idp.gast.it.uc3m.es:389
            binddn="cn=Replicator,dc=gast.it.uc3m,dc=es"
            bindmethod=simple credentials=secret
 replica uri=ldaps://idp.gast.it.uc3m.es:636
           binddn="cn=Replicator,dc=gast.it.uc3m,dc=es"
           bindmethod=simple credentials=secret
# indexed attribute definitions
   index uid pres,eq
   index cn,sn,uid pres,eq,approx,sub
   index objectClass eq
# database access control definitions
 access to attrs=userPassword
             by self write
         by anonymous auth
             by dn.regex="cn=admin,dc=gast.it.uc3m,dc=es" write
             by * none

 access to *
             by self write
             by dn.regex="cn=admin,dc=gast.it.uc3m,dc=es" write
             by * read
# Cleartext passwords, especially for the rootdn, should
# be avoid.  See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw      secret
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
```

Como se puede apreciar, este fichero consta de una parte de configuración global, en la cual indicamos aquellos esquemas que deseamos que siga OpenLDAP por medio de la directiva `include`. También se definen aquellos ficheros en los que se almacenarán el identificador de proceso (*pid*) y los argumentos con los que se ha llamado al programa, mediante las directivas `pidfile` y `argsfile`. En esta sección global se pueden definir además, listas de control de acceso (**ACL** en inglés, *Access Control List*) para la clave.

En la siguiente sección del fichero, nos encontramos con la definición de los parámetros relativos a la base de datos. Con la directiva `database` indicamos el tipo de base de datos a utilizar. La directiva `directory` define el directorio en el que se va a crear la base de datos. Aunque la información a guardar en este directorio se almacenará aparte, el fichero de configuración debe indicar algunos aspectos básicos de estructuración de la información. Es necesario especificar cuál es el sufijo del nodo raíz y cuál es el usuario administrador (*root*).

Para proporcionar el valor asociado al nodo raíz de la jerarquía de nuestro directorio LDAP, disponemos de la directiva `suffix`. Se debe poner un sufijo que se ajuste a la estructura de datos que vamos a almacenar. En el ejemplo mostrado, el sufijo sería:

```
suffix          "dc=gast.it.uc3m,dc=es"
```

Además, debemos crear un usuario que sea el administrador del directorio LDAP. Esto lo conseguimos mediante las siguientes líneas del fichero de configuración:

```
rootdn      "cn=admin,dc=gast.it.uc3m,dc=es"
rootpw      secret
```

La primera directiva crea un nombre distinguido para el usuario *root* y la segunda indica la clave de acceso. En el ejemplo mostrado, la clave se encuentra en modo texto plano, lo cual para realizar pruebas nos puede servir. Sin embargo, es aconsejable almacenarla de forma cifrada.

Otra directiva que cabe resaltar es `index`. Con ella podemos indicar qué búsquedas son las más habituales, con el fin de crear índices que permitan obtener respuestas más rápidas en las consultas. Respecto a la directiva `access`, sirve para indicar quién puede acceder al directorio LDAP y qué tipo de permisos tiene (lectura, escritura, etc.).

Una vez tenemos completada la definición del fichero de configuración de nuestro servidor LDAP, podemos proceder a lanzarlo mediante la introducción de la siguiente instrucción en la línea de comandos:

```
sudo /usr/local/libexec/sldap -f /usr/local/etc/openldap/
slapd.conf -d1 -h ldap://idp.gast.it.uc3m.es:389
```

Mediante la opción -f se especifica la ruta del fichero de configuración. Con -h se indica el nombre del *host* y número de puerto en el que se lanzará el servicio. Éste por defecto se lanzará en el puerto 389, pero se puede utilizar esta opción para abrir un puerto diferente. Con la opción -d se muestra por pantalla lo que está sucediendo en el servidor, lo cual nos puede resultar de gran utilidad en caso de que se produzcan errores, para poder determinar sus causas.

Para comprobar que nuestro servidor está funcionando correctamente, podemos introducir la siguiente instrucción en la línea de comandos:

```
ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
```

Si nuestro servidor está funcionando correctamente, la salida por consola debería presentar un aspecto similar al siguiente:

```
.....

dn:
namingContexts: dc=gast.it.uc3m, dc=es

.....
```

En este momento ya tenemos instalado y configurado el servidor basado en OpenLDAP correctamente. El siguiente paso, es añadir entradas al directorio LDAP. La siguiente sección está dedicada a explicar cómo podemos crear nuevos usuarios, modificar sus datos, comprobar si un usuario se encuentra registrado en nuestro directorio LDAP, etc.

B.3.3. Administración del directorio LDAP con phpLDAPadmin

Llegados a este punto, tenemos un servidor LDAP dispuesto a ofrecer información, pero está completamente vacío así que no hay información que ofrecer. Por tanto, el siguiente paso que debemos realizar es rellenar el directorio. Para ello, disponemos de un tipo de fichero llamado LDIF, que como comentamos en 2.4.2, es un archivo de texto que contiene la información que vamos a agregar al directorio y que nos permite preparar los datos para ser añadidos correctamente.

Para añadir o modificar estos datos, tenemos dos posibilidades: utilizar las instrucciones *ldapadd* y *ldapmodify* de OpenLDAP e introducirlas en la línea de

comandos; o disponer de un explorador de directorios LDAP que nos permitan realizar estas operaciones de una forma más sencilla e intuitiva. Existen diversos exploradores LDAP tanto de pago como libres. Entre las aplicaciones libres, podemos destacar ***gq***, ***phpldapadmin*** y ***JXplorer***. Nosotros hemos optado por ***phpldapadmin***, ya que podemos reutilizar la instalación del servidor web Apache para ponerla en funcionamiento.

Instalación de phpLDAPadmin A continuación, describimos los pasos necesarios para llevar a cabo la instalación y configuración de nuestro explorador de directorios:

- Si disponemos de permisos de administrador, con la ayuda del paquete *apt-get*, introducimos en la línea de comandos:

  ```
  sudo apt-get install phpldadmin
  ```

- En caso de no disponer de estos permisos, podemos descargarnos este paquete de la siguiente URL: `http://phpldapadmin.sourceforge.net/` y seleccionar un directorio para la ubicación de los ficheros fuentes y en el cual, se vaya a llevar a cabo el proceso de compilación e instalación. Los pasos a seguir, son los siguientes:

  ```
  ./configure
  make
  make install
  ```

  Al ejecutar *configure* podemos añadir preferencias de compilación e instalación, como por ejemplo especificar con la directiva `--prefix` el directorio en el que deseamos que se sitúen los ficheros binarios y de configuración.

Construcción del fichero de configuración Una vez que tenemos instalado nuestro explorador de directorios LDAP, el siguiente paso es configurar un fichero llamado *config.php*, que tras el proceso de instalación con permisos de administrador ha quedado situado en el directorio `/etc/phpldapadmin/`.

Por defecto la forma que utiliza la aplicación para autenticarse con el servidor de LDAP es manual, es decir, debemos introducir un usuario y una contraseña, pero existe también la posibilidad de especificar que la autenticación sea automática:

- Para que el explorador de directorios se conecte automáticamente al servidor LDAP, con un usuario `cn=admin,dc=gast.it.uc3m,dc=es` y contraseña `secret`, debemos definir las siguientes líneas en el fichero de configuración:

```
$ldapservers->SetValue($i,'server','auth_type','config');
$ldapservers->SetValue($i,'login','dn',
     'cn=admin,dc=gast.it.uc3m,dc=es');
$ldapservers->SetValue($i,'login','pass','secret');
$ldapservers->SetValue($i,'server','tls',false);
```

- Si por el contrario, deseamos que nos pida que introduzcamos un usuario y una clave para conectarnos al servidor LDAP, debemos definir las siguientes líneas en el fichero de configuración:

```
$ldapservers->SetValue($i,'server','auth_type','cookie');
$ldapservers->SetValue($i,'login','dn','');
$ldapservers->SetValue($i,'login','pass','');
$ldapservers->SetValue($i,'server','tls',false);
$config->custom->session['blowfish'] = "cadena";
```

  Cabe resaltar, que en este caso debemos dejar vacías las variables `dn` y `pass`. Por otro lado, la cadena `blowfish` puede ser cualquier cadena y se utiliza para encriptar la *cookie*.

Otras preferencias que se pueden indicar en este fichero de configuración son por ejemplo, el idioma o que se escriba en un fichero de *log* el registro de acciones llevadas a cabo, así como los posibles errores que se hayan podido producir. Para ello, en el caso de estos dos ejemplos concretos, debemos añadir la siguientes líneas:

```
$config->custom->appearance['language'] = 'es';
$config->custom->debug['level'] = 255;
$config->custom->debug['syslog'] = true;
$config->custom->debug['file'] = '/tmp/pla_debug.log';
```

En este ejemplo, el fichero de *log* se escribirá en el directorio `/tmp`, aunque cualquier otra ubicación podría ser válida para este propósito.

Una vez tenemos completada la definición del fichero de configuración *config.php*, podemos proceder a arrancar el servidor mediante la introducción de la siguiente instrucción en la línea de comandos:

```
sudo /etc/init.d/apache2 restart
```

Si introducimos en un navegador la siguiente URL: `http://idp.gast.it.uc3m.es/phpldapadmin/`, nos aparece la interfaz gráfica que se muestra en la figura B.1:

Figura B.1: Vista de la interfaz de autenticación de phpLDAPadmin

En nuestro caso, hemos definido en el fichero de configuración que la autenticación se realice de modo automático, por lo que el campo en el que se debe introducir el usuario se rellena automáticamente y simplemente deberemos introducir el valor de la contraseña que hemos definido en el archivo de configuración. Tras registrarnos correctamente en el servidor LDAP, podemos proceder a crear nuevas entradas en el servidor.

Creación de entradas en el servidor LDAP Existen varias formas de crear nuevos usuarios en el servidor. La primera de ellas es seleccionar en la interfaz gráfica la opción *Crear nuevo objeto*, tras lo cual se nos permitirá elegir entre una serie de plantillas que se muestran en la figura B.2.

En estas plantilla podemos encontrar objetos definidos en LDAP y comentados en 2.4.2 como por ejemplo, *poxisAccount* o *InetOrgPerson*, que nos pueden resultar de utilidad para definir los datos necesarios de una cuenta de usuario, u *Organisational Role*, que nos permite definir una organización en la base de datos, etc. Tras la elección de una plantilla, debemos rellenar los valores de los atributos

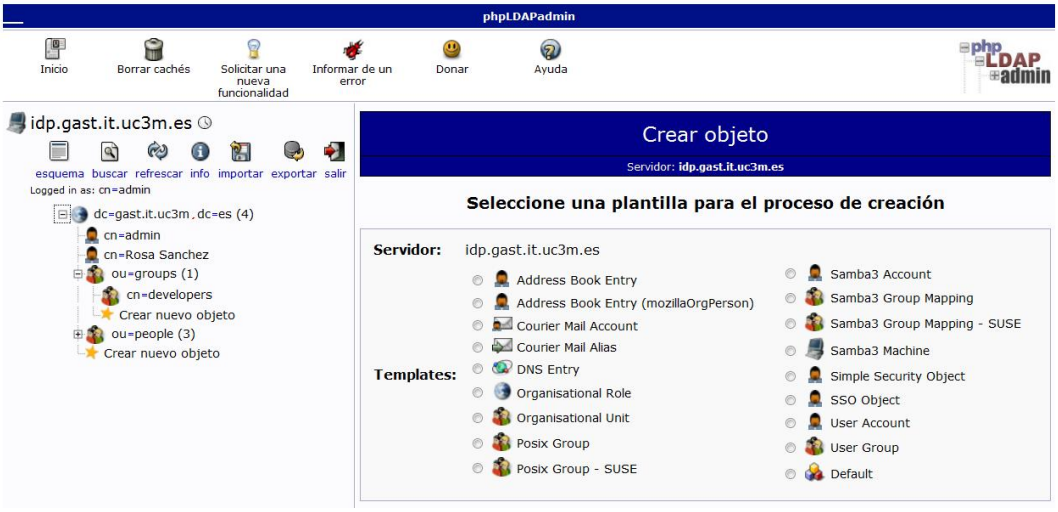

Figura B.2: Vista de las plantillas LDAP ofrecidas por phpLDAPadmin

del nuevo usuario. En función del tipo de plantilla seleccionada, habrá atributos opcionales, cuyo valor no es preciso indicar; y atributos obligatorios, que será necesario rellenar.

Otra manera de crear nuevos usuarios, es seleccionar en la interfaz de configuración la opción: `Crear nuevo objeto > importar`, la cual nos permitirá utilizar un fichero LDIF que hayamos definido previamente para crear una entrada.

En la figura B.3 se muestra un ejemplo de directorio LDAP en el que se ha definido un administrador: *admin*. También se ha creado un grupo llamado *people* que cuenta con tres usuarios. La estructura del árbol generado la podemos visualizar en la figura B.4. El tipo de objeto utilizado para crear estas tres entradas del grupo *people* es *userAuthentic* , el cual se trata de una nueva clase de objeto que definimos con el objetivo de que los atributos que lo forman se ajustasen a las necesidades de configuración LDAP del proveedor de identidad, como comentamos en 4.5. A continuación, se ilustra un ejemplo de fichero LDIF importado para crear el usuario `Rosa Sanchez` mostrado en la Figura B.3

```
dn: uid=rsanchez,ou=people,dc=gast.it.uc3m,dc=es
cn: Rosa Sanchez
objectClass: userAuthentic
sn: Sanchez
uid: rsanchez
uidNumber: 2001
gidNumber: 2000
```

Figura B.3: Ejemplo de directorio LDAP

```
homeDirectory: /usr/local/home/rsanchez
mail: rsanchez@it.uc3m.es
userPassword: rosa
```

Como se puede apreciar, en el nombre distinguido se especifica el grupo al que pertenece el usuario y un identificador del mismo. También se definen una serie de atributos, de los cuales, algunos de ellos tienen carácter obligatorio y otros, son opcionales, como comentamos en el capítulo 4.

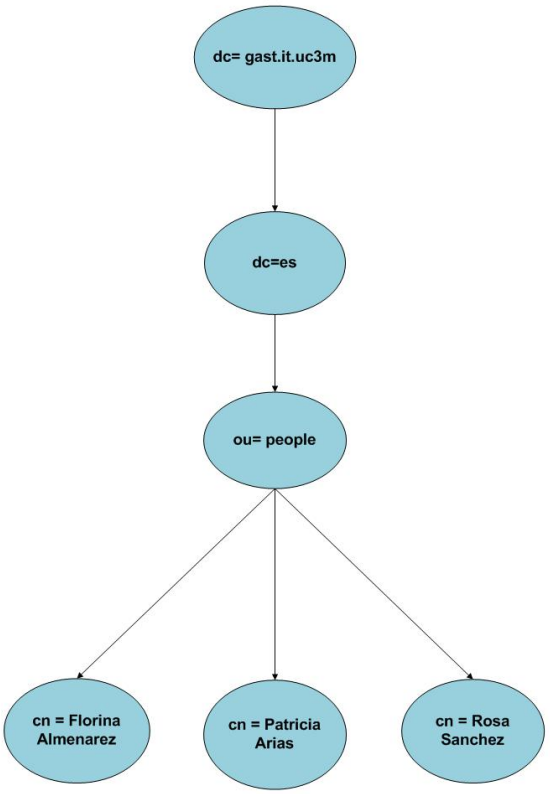

Figura B.4: Ejemplo de estructura de árbol LDAP

Apéndice C

Manual de generación de una Autoridad de Certificación

Durante el desarrollo de este proyecto, ha sido necesario disponer de certificados digitales, tanto para realizar la configuración de las entidades del sistema de gestión de identidad (SP e IdP) y de los servidores web con soporte SSL, a través de los cuales éstas ofrecen sus servicios; así como para la realización de la fase de pruebas. Para llevar a cabo la generación de estos certificados, hemos utilizado la librería criptográfica OpenSSL.

En esta sección se describe el procedimiento que se debe seguir para poner en funcionamiento una Autoridad de Certificación básica, que nos permita generar cuantos certificados deseemos, utilizando la herramienta de línea de comandos de OpenSSL.

C.1. Establecimiento del entorno

Para establecer el entorno de la Autoridad de Certificación (CA[1]) es necesario crear una serie de carpetas y ficheros, tal y como se describe a continuación.

1. En primer lugar, debemos escoger una ubicación para almacenar todos los archivos de la CA. En nuestro caso, hemos eligido la ruta `$HOME/ssl_test/demoCA`, aunque cualquier localización podría ser válida para este propósito.

[1]En adelante usamos indistintamente CA y AC (del inglés Certification Authority)

2. Una vez creado el directorio raíz, incluiremos en él cuatro nuevas carpetas, llamadas *certs*, *newcerts*, *crl* y *private*, que se detallan a continuación:

   - *certs*: Es el directorio donde se van a almacenar los certificados expedidos.
   - *newcerts*: Es el directorio donde se van a almacenar los certificados que acaban de ser firmados.
   - *crl* : Es el directorio donde se van a almacenar los certificados revocados.
   - *private*: Es el directorio donde se va a almacenar la clave privada de la CA.

3. Por último, se deben crear los siguientes ficheros:

   - *serial*: Este fichero sirve para registrar el número de serie de los certificados emitidos, de modo que no se permite la existencia de dos certificados con el mismo número de serie emitidos por la misma CA. Especifica el número de serie que tendrá el siguiente certificado que sea firmado. En cuanto al contenido del fichero, puesto que OpenSSL espera encontrar un número hexadecimal de al menos dos dígitos, escribiremos el valor 01.Este fichero indica el número de
   - *index.txt*: Este archivo deberá estar vacío inicialmente y funcionará como una base de datos de los certificados emitidos.
   - *openssl.cnf*: Este fichero contiene la información relativa acerca de la configuración de la CA. Sin embargo, la creación de este archivo de configuración requiere una complejidad mayor que los dos anteriores, por lo dedicamos la siguiente sección a describir detalladamente este proceso.

C.2. Creación del fichero de configuración de la CA

Para almacenar la información de configuración de una Autoridad de Certificación, es necesario crear un fichero que almacenaremos con el nombre *openssl.cnf*. A continuación se muestra el contenido que debe tener este archivo:

```
[ ca ]
default_ca  = CA_default        # The default ca section

####################################################################
```

```
[ CA_default ]

dir     = ./demoCA      # Where everything is kept
certs       = $dir/certs        # Where the issued certs are kept
crl_dir     = $dir/crl      # Where the issued crl are kept
database    = $dir/index.txt    # database index file.
new_certs_dir   = $dir/newcerts     # default place for new certs.

certificate = $dir/cacert.pem   # The CA certificate
serial      = serial        # The current serial number
crl     = $dir/crl.pem      # The current CRL
private_key = $dir/private/cakey.pem# The private key
RANDFILE    = $dir/private/.rand    # private random number file

x509_extensions = usr_cert      # The extentions to add to the cert

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions    = crl_ext

default_days    = 365           # how long to certify for
default_crl_days= 30            # how long before next CRL
default_md  = md5           # which md to use.
preserve    = no            # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy      = policy_match

# For the CA policy
[ policy_match ]
countryName     = match
stateOrProvinceName = match
organizationName    = match
organizationalUnitName  = optional
commonName      = supplied
emailAddress        = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName     = optional
stateOrProvinceName = optional
localityName        = optional
organizationName    = optional
organizationalUnitName  = optional
commonName      = supplied
emailAddress        = optional

####################################################################
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes      = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

[ req_distinguished_name ]
countryName         = Country Name (2 letter code)
countryName_default     = ES
```

```
countryName_min         = 2
countryName_max         = 2

stateOrProvinceName     = State or Province Name (full name)
stateOrProvinceName_default = Madrid

localityName            = Leganes

0.organizationName      = Organization Name (eg, company)
0.organizationName_default  = UCIIIM

# we can do this but it is not needed normally :-)
#1.organizationName     = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName      = Organizational Unit Name (eg, section)
organizationalUnitName_default  = IT

commonName          = Common Name (eg, YOUR name)
commonName_default      = Andres Marin Lopez
commonName_max          = 64

emailAddress            = amarin@it.uc3m.es
emailAddress_max        = 40

# SET-ex3           = SET extension number 3

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min       = 0
challengePassword_max       = 20

unstructuredName        = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:TRUE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType            = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
```

```
nsComment           = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy

# Copy subject details
# issuerAltName=issuer:copy
```

El comando OpenSSL para realizar funciones relacionadas con una Autoridad de Certificación es `ca`, tal y como se muestra en la primera sección del fichero de configuración. Como podemos observar, el contenido de esta sección es una única instrucción: `default_ca`. Su valor es el nombre de la siguiente sección del archivo, donde se definen los parámetros de configuración por defecto de la CA. OpenSSL permite incluir varias configuraciones diferentes en el mismo fichero. En caso de no especificar el nombre de la configuración que se desea utilizar, se empleará aquélla indicada por `default_ca`.

Si observamos las diez primeras líneas de configuración, vemos que sirven para indicar dónde se encuentran los directorios y ficheros que constituyen el entorno de la CA. Las dos líneas que siguen, `default_days` y `default_md`, permiten indicar el número de días de validez de los certificados emitidos y el algoritmo de *hash* que se utilizará en las firmas respectivamente.

La directiva `policy` especifica el nombre de la sección en la que se define la política por defecto de la AC. La definición de la política constará de una serie de directivas cuyos nombres coinciden con los campos del DN o Distinguished Name de un certificado. Para cada una de estas directivas existen tres valores posibles: `match`, `supplied` y `optional`. El valor `match` significa que el contenido del campo con ese nombre debe ser el mismo en los certificados emitidos y en el certificado de la CA. El valor `supplied` indica que los certificados emitidos deben contener ese campo. Por último, el valor `optional` significa que el campo no es obligatorio.

Por otro lado, la última sección del documento contiene las extensiones que serán añadidas a todos los certificados expedidos por la CA. El nombre de esta sección viene especificado previamente por la directiva `x509_extensions`. En el caso de que esta directiva no estuviese presente en el fichero de configuración, OpenSSL generaría certificados con formato x509v1. Sin embargo, si esta directiva existe, los certificados creados son de tipo x509v3. En este ejemplo sólo añadiremos la extensión `basicConstraints`, la cual recibirá el valor `CA:false`, para indicar que los certificados emitidos por la CA no puedan ser utilizados a

su vez como certificados de CA.

C.3. Creación de un certificado raíz autofirmado

Para poder obtener un certificado propio que permita firmar los certificados expedidos, es necesario añadir los siguientes líneas en el fichero de configuración de la CA:

```
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes      = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

[ req_distinguished_name ]
countryName         = Country Name (2 letter code)
countryName_default     = ES
countryName_min         = 2
countryName_max         = 2

stateOrProvinceName     = State or Province Name (full name)
stateOrProvinceName_default = Madrid

localityName            = Leganes

0.organizationName      = Organization Name (eg, company)
0.organizationName_default  = UCIIIM

# we can do this but it is not needed normally :-)
#1.organizationName     = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName      = Organizational Unit Name (eg, section)
organizationalUnitName_default  = IT

commonName          = Common Name (eg, YOUR name)
commonName_default      = Andres Marin Lopez
commonName_max          = 64

emailAddress            = amarin@it.uc3m.es
emailAddress_max        = 40

# SET-ex3           = SET extension number 3

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min       = 0
challengePassword_max       = 20

unstructuredName        = An optional company name
```

```
[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:TRUE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType            = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment           = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy

# Copy subject details
# issuerAltName=issuer:copy
```

Como se puede apreciar, hemos incluido una sección para indicar qué opciones debe tomar el comando `req` cuando sea ejecutado. La directiva `default_bits` indica que la clave privada debe tener una longitud de 1024 bits. Si no se especifica ningún valor, la longitud por defecto sería de 512 bits, lo que supone una menor seguridad. Las directivas `default_keyfile` y `default_md`, permiten indicar la ubicación de la clave privada y el algoritmo que se empleará para firmar dicha clave.

Por otro lado, en el fichero de configuración podemos encontrar otra sección importante llamada `distinguished_name`, de la cual, OpenSSL obtendrá la información necesaria para rellenar los certificados.

Finalmente, la directiva `x509_extensions` especifica el nombre de una sección que incluye las extensiones que queremos añadir al certificado emitido. En este

caso, fijamos el valor de `ca` a `true` para que el certificado pueda ser utilizado como certificado de CA.

Una vez terminada la configuración de este fichero, ya podemos poner en funcionamiento la Autoridad de Certificación y generar nuestro propio certificado autofirmado mediante la siguiente instrucción de OpenSLL:

```
#openssl req -x509 -config HOME/ssl_test/openssl.cnf -newkey rsa: -out HOME/ssl_test/cacert.pem
-outform PEM
```

La salida por pantalla que resulta de la ejecución de este comando es:

```
Making CA certificate ...
Generating a 2048 bit RSA private key
..............+++
.............................+++
writing new private key to './demoCA/private/./cakey.pem'
-----
```

OpenSSL pide la introducción de una contraseña para cifrar la clave privada. Es importante elegir una contraseña segura para no poner en peligro la integridad de nuestra AC. La clave privada será cifrada utilizando el algoritmo 3DES con una clave derivada de la contraseña introducida.

A continuación se muestra el comando que ha de ejecutarse para visualizar el contenido del certificado autofirmado que acabamos de crear, junto con la salida obtenida:

```
#openssl x509 -in /HOME/ssl_test/demoCa/cacert.pem -inform PEM -text -noout

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            e8:2d:71:05:ea:0d:f5:f1
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ES, ST=Madrid, O=IT-UC3M, CN=Andres Marin
        Validity
            Not Before: Oct 22 22:37:40 2009 GMT
            Not After : Oct 22 22:37:40 2010 GMT
        Subject: C=ES, ST=Madrid, O=IT-UC3M, CN=Andres Marin
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:c0:dc:ef:9f:db:3e:3f:52:89:48:ec:b2:fc:92:
                    da:28:de:d7:30:4f:47:85:70:dc:48:76:65:23:fa:
                    59:f3:ac:16:71:d0:eb:0c:71:d9:37:88:f6:4c:33:
                    ed:45:d5:b1:e6:ee:52:7f:fc:af:6c:f3:2a:02:b7:
                    22:89:b7:1b:b9:e3:29:5c:b5:7e:f1:c6:d8:7e:2e:
                    15:6a:b6:76:2d:cd:be:67:97:b5:f0:aa:dd:0d:07:
```

```
                4f:8d:7e:d1:7b:c7:c1:20:e6:2e:dc:f6:55:b8:0a:
                5a:19:9b:54:fc:f4:13:a3:ea:a5:46:e8:46:c8:0d:
                1e:1f:fd:22:e1:13:63:6c:f8:63:dd:1b:78:a0:c4:
                97:fc:cf:a6:a0:f3:df:20:51:3f:0c:71:1c:fa:d1:
                6d:46:c5:1e:09:3c:24:20:79:b2:3b:1e:d7:83:fa:
                58:fb:21:0b:bc:e1:7c:5b:c3:ae:f0:5b:a7:76:2b:
                90:9b:45:f5:26:05:87:73:4a:c3:a8:61:60:88:09:
                d7:d7:8b:e7:ef:1a:dc:fa:f7:5f:19:74:eb:df:97:
                2e:f4:fd:28:2b:7c:4b:3a:8b:f4:6f:90:6e:28:fe:
                5b:75:e5:e2:c8:2a:25:3a:59:80:22:06:b1:09:ea:
                31:0d:85:73:85:3d:1d:be:89:68:90:6d:d1:b7:4a:
                66:5d
            Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        41:4f:58:c3:0a:41:d8:18:d8:94:23:62:50:d5:33:39:87:4f:
        a3:34:36:3d:03:47:e0:74:f2:65:06:37:86:f0:28:00:bf:47:
        57:23:bc:bb:dc:45:24:e3:f6:18:a7:ff:d3:f7:51:15:b2:64:
        7e:0a:ba:ce:8e:bd:da:96:6b:77:3e:70:dc:22:ee:ec:89:a3:
        2a:a3:77:b4:75:8c:5a:18:ac:e4:6c:d6:da:ce:51:ae:7b:74:
        a3:46:70:6a:02:33:9a:4e:69:42:1b:f2:e7:66:bd:f0:38:0c:
        c7:19:0f:86:fb:30:98:00:c5:87:7c:4f:e0:45:7c:21:38:e5:
        d8:41:c9:7f:75:a0:7f:8c:8f:1f:3e:18:45:b4:20:14:ce:ba:
        9c:9b:07:7c:8b:60:f5:29:c6:1b:b8:d7:1f:64:6c:f5:9b:36:
        5b:7d:27:ef:37:16:b5:71:40:ef:e2:c1:5c:90:89:60:02:6e:
        73:d6:d9:d3:fa:34:34:0a:e9:1c:f9:9c:57:c2:c7:d6:fe:c5:
        99:95:50:bd:f2:d6:b8:0f:8b:94:98:b8:5e:30:d7:97:41:0d:
        49:8f:d4:bb:41:23:70:94:fe:65:3f:07:5b:7f:2c:50:4c:9f:
        61:d5:b2:9f:ec:f0:9f:bd:f7:6e:eb:72:32:52:c8:31:96:ef:
        04:fd:97:44
```

C.4. Generación de certificados

Una vez hayamos completado todos los pasos descritos en los apartados anteriores, nuestra AC estará en disposición de emitir certificados digitales. Para hacer esto, necesitamos peticiones de certificado válidas, que pueden generarse ejecutando la instrucción `req` por línea de comandos. Con el fin de facilitar el proceso, se ha realizado un *script* llamado *crea.sh* que contiene las instrucciones necesarias para generar un certificado. A continuación mostramos el *script* mencionado:

crea.sh

```
#!/bin/sh
mihost="$HOSTNAME"
mihost_key=$mihost"_key"
mihost_req=$mihost"_req"
mihost_cert=$mihost"_cert"
sh testca -ca
openssl req -config client.cnf -new -keyout
        $mihost_key -out $mihost_req.pem -days 365
openssl ca -config openssl.cnf -verbose -cert demoCA/cacert.pem
        -policy policy_match -out $mihost_cert.pem -infiles $mihost_req.pem
```

Por tanto, para generar un certificado válido, basta con introducir en la línea de comandos la instrucción:

```
./crea.sh
```

A continuación se muestra la salida obtenida por consola:

```
Generating a 1024 bit RSA private key
...................++++++
...................++++++
writing new private key to '_key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [ES]:ES
Province Name [Madrid]:Madrid
Common Name (eg, YOUR name) []:ZXID SP
Common Name (eg, YOUR name) []:http://sp.gast.it.uc3m.es
Organization Name (eg, company) []:IT-UC3M
rsanchez@inv.it.uc3m.es []:
server, email, objsign []:
```

Como resultado del comando `openssl req -config client.cnf -new -keyout $mihost_key -out $mihost_req.pem -days 365`, que figura en el *script*, se crean dos nuevos ficheros: *mihost_req.pem* y *mihost_key*. El primero de ellos contiene la petición de certificado, mientras que el segundo contiene la clave privada asociada a la clave pública incluida en la petición.

Una vez que disponemos de una petición, ya podemos usar nuestra AC para emitir el certificado digital solicitado. Esto se consigue gracias al comando `openssl ca -config openssl.cnf -verbose -cert demoCA/cacert.pem -policy policy_match -out $mihost_cert.pem -infiles $mihost_req.pem`, que figura también en el *script crea.sh*. En este caso, la salida obtenida por consola es la siguiente:

```
Using configuration from openssl.cnf
0 entries loaded from the database
generating index
message digest is md5
policy is policy_match
next serial number is 10
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=ES, ST=Madrid, CN=ZXID SP, CN=http://sp.gast.it.uc3m.es, O=IT-UC3M
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
```

```
                Modulus (1024 bit):
                    00:c7:da:50:45:66:8d:40:83:a8:f7:fd:88:3f:cc:
                    0d:3c:29:8c:65:1c:1b:43:a1:81:c6:16:95:55:c4:
                    aa:40:6f:64:da:3e:93:d6:99:5e:24:d0:0f:6f:d3:
                    f4:77:da:0d:82:ae:01:d1:c1:dc:52:1e:37:9c:8f:
                    57:fd:6a:32:11:fe:cd:b8:54:80:30:cd:a7:6c:4b:
                    2d:45:3c:1c:ff:bb:95:65:77:22:af:2d:b0:d6:59:
                    08:0a:46:3f:a9:de:3f:13:7e:d7:89:e8:cb:76:4d:
                    ea:33:cb:b6:bf:38:5d:84:c1:d8:90:d2:00:16:62:
                    1a:8e:63:26:a6:7b:58:f2:05
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        7f:90:d7:a9:ed:ed:f5:49:82:6f:93:11:4d:a9:e0:65:b5:84:
        99:b8:04:af:2c:92:7e:5f:86:10:c5:a4:ba:71:bd:af:bf:3d:
        17:6f:51:54:5d:56:68:e5:bc:7e:6f:3a:2c:70:31:d7:89:0b:
        f4:f5:39:66:9e:30:8e:ec:e4:70:53:46:ef:ae:d9:7d:aa:f3:
        14:f7:2f:c3:a7:27:d1:13:2d:9d:6c:79:f1:1c:a8:da:c6:fc:
        ee:3c:fb:16:11:d9:dc:ef:40:e2:8c:d2:25:83:13:db:eb:19:
        ba:f1:28:45:35:14:ee:97:a7:90:aa:76:63:90:e2:5b:de:8c:
        2e:34
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'ES'
stateOrProvinceName   :PRINTABLE:'Madrid'
commonName            :PRINTABLE:'ZXID SP'
commonName            :PRINTABLE:'http://sp.gast.it.uc3m.es'
organizationName      :PRINTABLE:'IT-UC3M'
The subject name appears to be ok, checking data base for clashes
Everything appears to be ok, creating and signing the certificate
Successfully added extensions from config
Certificate is to be certified until Oct 23 11:29:30 2010 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
writing new certificates
writing ./demoCA/newcerts/10.pem
Data Base Updated
```

Lo primero que sucede es que OpenSSL pide al usuario una contraseña. Esta contraseña no corresponde a la clave privada de la petición de certificado sino que se trata de la correspondiente a la clave privada de la AC. Dicha clave será utilizada para firmar el nuevo certificado, siempre y cuando esta operación sea confirmada. Finalmente, OpenSSL pregunta si se desea actualizar la base de datos de la AC y muestra por pantalla el contenido del certificado expedido.

El último aspecto a comentar es que el certificado resultante también se almacena en el directorio que habíamos especificado anteriormente en el fichero de configuración, es decir, en la carpeta *$HOME/ssl_test/demoCA/certs*. Dicho certificado recibe un nombre consistente en el número de serie del mismo unido a la extension *.pem*, que en el caso del ejemplo mostrado, se llama *10.pem* .

Apéndice D

Glosario de términos

Apache Es un servidor web HTTP/HTTPS de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, etc. Este servidor se desarrolla dentro del proyecto HTTP *Server* (httpd) de la Apache Software Foundation. Ofrece las ventajas de ser altamente configurable, presentar bases de datos de autenticación, negociado de contenido, etc.

Asertos (en inglés, *Assertions*) Es una colección de una o más afirmaciones o declaraciones acerca de una entidad, por ejemplo, una declaración de autenticación o una declaración de autorización. También puede definirse como la información de identidad proporcionada por un IdP a un SP.

Atributo Es aquella información asociada con una entidad que especifica una característica de la misma y que permite describirla. En un sistema de gestión de identidad, objetos y clases de objetos están compuestos de atributos.

Autorización Conceder acceso basado en derechos de acceso.

Authentic Es un proveedor de identidad en código abierto basado en la librería Lasso.

Certificado Es un conjunto de datos relevantes de seguridad emitidos por una autoridad de seguridad o una tercera parte de confianza, junto con información de seguridad que son utilizados para proporcionar servicios de integridad y autenticación del origen.

Círculo de confianza Es un conjunto de criterios establecidos para unir organizaciones dentro de una federación con el fin de acceso fiable a los recursos de cada otro.

Confianza Es una valoración subjetiva que permite que una entidad esté dispuesta a recurrir a otra entidad para ejecutar un conjunto de acciones y/o hacer un conjunto de asertos acerca de una serie de temas y/o ámbitos.

COIT Colegio Oficial de Ingenieros de Telecomunicación.

Control de acceso Es la prevención del uso de un recurso de manera no autorizada.

Contexto Es una propiedad que puede ser asociada con un atributo de usuario para especificar información que puede ser utilizada para determinar la aplicabilidad del valor.

Defederación Es un procedimiento que permite indicar que un identificador persistente que ha sido previamente establecido en un proceso de federación, dejará de ser utilizado. Es decir, se elimina el acuerdo entre dos proveedores para referirse a un usuario.

Descubrimiento Es un proceso por el cual recursos de un sistema de gestión de identidad pueden ser encontrados o localizados.

DN *Distinguished Name* . Es un nombre único que se compone de diversos campos, siendo los más frecuentes los siguientes: CN (*Common Name*), OU (*Organizational Unit*), O (*Organization*) y C (*Country*).

Entidad Es alguien o algo (un sistema *software*, un objeto, un dispositivo, una persona, etc.) que se caracteriza a través de la medida de sus atributos.

Federación Este término es usado en dos sentidos: 1) el acto de establecer una relación productor-consumidor entre dos o más entidades/reinos, 2) una asociación abarcando un número de proveedores de servicios y de identidad.

Firma digital Es un proceso utilizado para adjuntar un código digital único de quién firma a un mensaje electrónico. Una firma digital resulta del uso de una clave privada para firmar un mensaje. El receptor del mensaje puede usar la clave pública de quién firmó el mensaje para verificar si la firma digital es válida, y si el mensaje ha sido modificado desde que fue firmado.

HTTP *HyperText Transfer Protocol*. Es un protocolo de transferencia de hipertexto, orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

Identidad federada Una identidad local de un usuario/principal única que puede ser utilizada para acceder a un grupo de servicios o aplicaciones que están delimitadas por los vínculos y condiciones de una federación.

IPC Índice de Precios de Consumo. Es un promedio ponderado de los precios de los bienes y servicios consumidos por las familias, como por ejemplo el calculado en España por el Instituto nacional de Estadística.

LDAP Protocolo Ligero de Acceso a Directorios (en inglés, *Lightweight Directory Access Protocol*). Es un protocolo que permite el acceso a un servicio de directorio ordenado y distribuido para introducir, buscar o modificar diversa información en un entorno de red de manera eficiente.

Lasso *Liberty Alliance Single Sign-On*. Es una librería implementada en C, desarrollada por una compañía francesa llamada *Entrouvert*, con *bindings* para distintos lenguajes como Java, Perl, Python y PHP. Esta librería soporta Liberty ID-FF 1.2, ID-WSF y SAMLv2.

LibNeon Es una librería escrita en C, de código abierto, que implementa los protocolos HTTP y HTTPS.

Metadato (en inglés, *Metadata*) Es un documento XML que contiene información sobre configuración (por ejemplo, perfiles SAML/Liberty soportados por una entidad) y material de seguridad (claves, certificados digitales, etc.) y tiene como propósito facilitar el despliegue de sistemas SAML/Liberty. La información contenida en el metadato puede estar asociada a una sola entidad o bien a un grupo de entidades.

OASIS *Organization for the Advancement of Structured Information Standards*. Es una organización sin ánimo de lucro que impulsa el desarrollo, la convergencia y la adopción de estándares abiertos para la sociedad de la información mundial. Fundada en 1993, OASIS tiene más de 5.000 participantes, que representan más de 600 organizaciones y miembros individuales en 100 países.

OpenSSL El software OpenSSL es un proyecto de *software* libre desarrollado por los miembros de la comunidad Open Source. Se trata de un robusto juego de herramientas que ayudan a su sistema a implementar el protocolo *Secure Sockets Layer* (SSL), así como otros protocolos relacionados con la seguridad, tales como el *Transport Layer Security* (TLS). También incluye una librería de criptografía.

OpenLDAP Es una implementación de código abierto del protocolo LDAP, desarrollada desarrollado por los miembros de la comunidad Open Source. Este software funciona en diversas plataforma como Linux, Mac OS X, Solaris, Microsoft Windows (NT y derivados, incluyendo 2000, XP, Vista), etc.

Perfil (*profile*) Es un conjunto de reglas definidos para uno o varios fines; a cada conjunto se le asigna un nombre en el patrón "*xxx profile of SAML*" o "*xxx SAML profile*". Se definen reglas por ejemplo, para saber cómo incrustar asertos y extraerlos de un protocolo u otro contexto de uso, para para el uso de mensajes del protocolo SAML en un contexto particular de uso, etc.

Phishing Es una modalidad de estafa por correo electrónico diseñada con la finalidad de robarle la identidad a un usuario, con el fin de adquirir información confidencial.

PKI *Public Key Infrastructure*. Es una combinación de *hardware* y *software*, políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas tales como el cifrado, la firma digital o el no repudio de transacciones electrónicas.

Principal Es una entidad del sistema cuya identidad puede ser autenticada.

IdP Proveedor de Identidad (en inglés, *Identity Provider*). Es una entidad que emite, mantiene y gestiona una identidad digital fiable para otras entidades.

Pseudónimo Es una identidad ficticia que una entidad crea por sí misma, en virtud de la cual la entidad puede permanecer anónima en ciertos contextos.

SASL *Simple Authentication and Security Layer*. Es un estándar que separa los mecanismos de autenticación de los protocolos de la aplicación, de tal forma que, cualquier protocolo de aplicación que use SASL, puede utilizar cualquier mecanismo de autenticación soportado por SASL. Los mecanismos basados en este protocolo se modelan como una sucesión de retos y respuestas.

SLO *Single logout*. Registro único de salida, es decir, un usuario sale o se desregistra con una sola acción de todos los servicios en los que se había registrado.

SSO *Single Sign-On*. Es un proceso en el cual se autentica el usuario mediante una única acción (normalmente frente al IdP) que le será válida para los distintos servicios federados registrados en los que tiene cuenta el usuario durante la vida de la sesión.

SOAP Protocolo Simple de Acceso a Objetos (en inglés, *Simple Object Access Protocol*). Es un protocolo que especifica cómo dos objetos en diferentes procesos pueden comunicarse por medio del intercambio de datos XML.

SSL *Secure Sockets Layer*. Es un protocolo criptográfico que proporciona comunicaciones seguras en Internet.

SWIG *Simplified Wrapper and Interface Generator*. Es una herramienta de código abierto que se utiliza para conectar los programas o bibliotecas escritas en C/C++ con lenguajes de alto nivel como Perl, Python, Ruby, PHP, Java, C#, etc.

UIT-T Sector de Normalización de las Telecomunicaciones de la UIT. Es el órgano permanente de la Unión Internacional de Telecomunicaciones (UIT) que estudia los aspectos técnicos, de explotación y tarifarios y publica normativa sobre los mismos, con vista a la normalización de las telecomunicaciones a nivel mundial.

X.509 X.509 es un estándar UIT-T para infraestructuras de claves públicas. Especifica, entre otras cosas, formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación.

XML *Extensible Markup Language*. Es un lenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C) que proporciona una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son SAML, XHTML, etc.

ZXID Es una librería escrita en C que soporta el rol del SP en la versión 2.0 de SAML. También soporta otros protocolos como ID-FF, ID-WSF y WS-Fed.

Bibliografía

[1] Página web oficial de OASIS. Disponible en `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`. Noviembre, 2009.

[2] Security Assertion Markup Language (SAML) V2.0, Technical Overview Committee Draft 02, 25 March 2008. Disponible en `http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf`

[3] R. Aarts, J. Beatty, C. Cahill, X. Serret, G. Whitehead. Liberty ID-FF Protocols and Schema Specification, version 1.2. En Liberty Alliance Project. Mayo 2005.

[4] S. Cantor, J. Hodges, J. Kemp, P. Thompson. Liberty ID-FF Architecture Overview, version 1.2.

[5] J. Tourzan, Y. Koga (eds.). Liberty ID-WSF Web Services Framework Overview, Version: 2.0. Liberty Alliance Project. Julio, 2007.

[6] Página oficial de la iniciativa Internet2. Agosto, 2009. Disponible en `http://www.internet2.edu`

[7] Página oficial de la iniciativa Shibboleth. Agosto, 2009. Disponible en `http://shibboleth.internet2.edu/`

[8] B. Fitzpatrick, D. Recordon, D. Hardt, J. Hoyt, J. Bufu (ed.). OpenID Authentication 2.0 -Final. Diciembre 2007

[9] M. Goodner, A. Nadalin (eds.). Web Services Federation Language (WS-Federation) Version 1.2, OASIS Standard. Mayo, 2009

[10] Whitfield Diffie and Martin Hellman. New Directions in Criptography. November, 1976.

[11] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, IETF Network Working Group, Mayo 2008.

[12] International Telecomunication Union (ITU-T). Recommendation X.509, The directory: Public Key and attribute certificate frameworks Series X: Data Networks and Open System Communications. Marzo, 2000.

[13] R. Housley , W. Ford, W. Polk & D.Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF Network Working Group. RFC 3280. April, 2002.

[14] International Telecomunication Union (ITU-T). Recommendation X.500, The Directory: Overview of Concepts, Models and Service. Agosto, 2005.

[15] Open Mobile Alliance (OMA). Enabler Release Definition for IMPS, Approved Version 1.3. Enero 2007.

[16] P. Mockapetris. Domain Names - Implementation and Specification, RFC 1035, IETF Network Working Group. Noviembre, 1987.

[17] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (LDAP), v3, RFC 2251, IETF Network Working Group. Diciembre, 1997.

[18] XML Signature Syntax and Processing (Second Edition), W3C Recommendation 10 June 2008. Disponible en `http://www.w3.org/TR/xmldsig-core/`

[19] Breve Guía de Seguridad, Recomendación W3C 7 Febrero 2008. Disponible en `http://www.w3c.es/divulgacion/guiasbreves/Seguridad`

[20] SOAP Version 1.2,W3C Recommendation (Second Edition) 27 April 2007. Disponible en `http://www.w3.org/TR/soap12`

[21] S. Cantor, J. Kemp, R. Philpott, E. Maler (eds.). Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard. Marzo 2005.

[22] ZXID: Open SAML implementation in C. Septiembre, 2009. Disponible en `http://www.zxid.org/`

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1, RFC 2616, IETF Network Working Group. Junio, 1999. En Liberty Alliance Project. Mayo 2005.

[24] Lasso: Liberty Alliance Single Sign-On. Octubre, 2009. Disponible en `http://lasso.entrouvert.org/`

[25] J. Hodges, J. Kemp, R. Aarts, G. Whitehead, P. Madsen (eds.). Liberty ID-WSF SOAP Binding Specification, Version: 2.0. Liberty Alliance Project. Julio, 2007.

[26] E. Rescorla. Diffie-Hellman Key Agreement Method, RFC 2631, IETF Network Working Group. Junio, 1999.

[27] SAML Open Source Implementations. Septiembre, 2009. Disponible en `http://saml.xml.org/wiki/saml-open-source-implementations`

[28] Libraries OpenID. Septiembre, 2009. Disponible en `http://wiki.openid.net/Libraries`

[29] D. Recordon, M. Jones, J. Bufu, J. Daugherty, N. Sakimura. OpenID Provider Authentication Policy Extension - Draft 5. Octubre, 2008.

[30] H. Lockhart, S. Andersen, J. Bohren, Y. Sverdlov, M. Hondo, H. Maruyama, A. Nadalin (ed.), N. Nagaratnam, T. Boubez, K. Morrison, C. Kaler (ed.), A. Nanda, D. Schmidt, D. Walters, H. Wilson, Ll. Burch, D. Earl, S. Baja, H. Prafullchandra. Web Services Federation Language (WS-Federation), Version 1.1. Diciembre 2006.

[31] Open WS-Federation implementation based on OpenSAML. Septiembre, 2009. Disponible en `http://code.google.com/p/open-wsfed11/`. Octubre, 2009.

[32] J. Hodge, R. Morgan. Lightweight Directory Access Protocol (v3): Technical Specification, RFC 3377, IETF Network Working Group. Septiembre, 2002.

[33] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3). Attribute Syntax Definitions, RFC 2252, IETF Network Working Group. Diciembre, 1997.

[34] M. Wahl, H. Alvestrand, J. Hodges, R. Morgan. Authentication Methods for LDAP, RFC 2829, IETF Network Working Group. Mayo, 2000.

[35] J. Myers. Simple Authentication and Security Layer, RFC 2222, IETF Network Working Group. Octubre, 1997.

[36] Página Web Oficial del Lenguaje de programación Python. Disponible en `http://www.python.org/`

[37] D. Robinson, K. Coar. The Common Gateway Interface (CGI) Version 1.1, RFC 3875, IETF Network Working Group. Octubre, 2004.

[38] Open SAML implentation in Java/C++. Agosto, 2009. Disponible en http://www.opensaml.org/

[39] Authentic: Liberty-compliant Identity Provider. Octubre, 2009. Disponible en http://authentic.labs.libre-entreprise.org/

[40] Open source implementation of the Lightweight Directory Access Protocol. Septiembre, 2009. Disponible en http://www.openldap.org/

[41] Web-based LDAP client. Septiembre, 2009. Disponible en http://phpldapadmin.sourceforge.net/wiki/index.php/Main-Page

[42] SAML implementation in Java. Septiembre, 2009. Disponible en http://opensso.org/

[43] SAML implementation in .NET. Septiembre, 2009. Disponible en http://xml.coverpages.org/SSO-SAML-DanishToolkits.html

[44] SAML and XACML implementation. Septiembre, 2009. Disponible http://www.esoeproject.org/

[45] SAML implementation in php. Septiembre, 2009. Disponible http://rnd.feide.no/simplesamlphp

[46] SAML v1.1 implementation. Septiembre, 2009. Disponible en http://www.sourceid.org/download/

[47] HTTP and WebDAV client library, with a C interface. Septiembre, 2009. Disponible en http://www.webdav.org/neon/

[48] Development Kit for C/C++ network programming that includes CGI/FastCGI supports. Noviembre, 2009. Disponible en http://www.qdecoder.org/

[49] Lasso Reference Manual, LassoProfile. Octubre, 2009. Disponible en http://lasso.entrouvert.org/documentation/api-reference/LassoProfile.html

[50] Lasso Reference Manual, LassoSamlp2AuthnRequest. Octubre, 2009. Disponible en http://lasso.entrouvert.org/documentation/api-reference/lasso-LassoSamlp2AuthnRequest.html.

[51] Almenárez, F., Arias, P., Marin, A.: Estado de la cuestión en seguridad e identidad. E.6.1.1 del Proyecto CENIT España Virtual. Octubre, 2008.

[52] Almenárez, F., Arbona, T., Arias, P., Blanco, J., Escalas, M., Marin, A., Sánchez, R.: Informe de avance en la investigación en "Seguridad e Identidad". H2 del Proyecto CENIT España Virtual. Mayo, 2009.

[53] Página web de información del IdP de Google basado en SAML. Octubre, 2009. Disponible en `http://sites.google.com/site/oauthgoog/fedlogininterp/saml-idp`

[54] Página web del COIT. Octubre, 2009. Disponible en `http://www.coit.es`

[55] Blog Financiero "De finanzas". Octubre, 2009. Disponible en `http://definanzas.com/2009/01/23/ipc-2009/`