Universitat Oberta de Catalunya

uoc.edu

Análisis de sistemas de autenticación y autorización para entornos web distribuidos

Oriol Parra Boldú
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Identidad Digital

Víctor Méndez Muñoz
Víctor García Font

31/12/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis de sistemas de autenticación y autorización para entornos web distribuidos*
Nombre del autor:	*Oriol Parra Boldú*
Nombre del consultor/a:	*Víctor Méndez Muñoz*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	12/2019
Titulación:	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Identidad Digital*
Idioma del trabajo:	*Castellano*
Palabras clave	*Sistemas distribuidos, identidad digital, identidad federada*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

En los últimos años se ha producido una importante evolución en la seguridad, capacidad y escalabilidad de los sistemas distribuidos debido a la necesidad de compartir recursos entre diferentes sistemas. Parte de estos recursos deben compartirse de forma segura debido a que contienen datos sobre la identidad de los potenciales usuarios de los sistemas, cobrando una gran importancia los sistemas de gestión de identidad federada.

En el presente trabajo, se han analizado los diferentes estándares de autenticación y autorización. Estos estándares son utilizados por los sistemas de gestión de identidad federada para compartir información de los usuarios garantizando su privacidad.

Posteriormente, para profundizar en los conceptos estudiados, se ha diseñado e implementado un sistema de autenticación y autorización utilizando los estándares OAuth 2.0 y *tokens* JWT. De esta forma, se propagará la identidad y los privilegios de los usuarios del sistema sin exponer sus credenciales.

Finalmente, se ha simulado la compartición segura de recursos distribuidos entre diferentes sistemas para realizar un estudio de la seguridad que ofrece el sistema diseñado.

Abstract (in English, 250 words or less):

In the last few years there's been an important evolution regarding security, capacity and scalability of the distributed systems due to the necessity to share resources amongst different systems. Part of these resources must be shared in a secured way as they contain data regarding the identity of potential system users, especially management of federal identity systems.

This study analyses the different standards of authentication and authorization. These standards are used by the Management of Federal Identity System in order to share the user information whilst guaranteeing their privacy.

Further to this analysis, a system of authentication and authorization used on OAuth 2.0 and tokens JWT standards has been designed and implemented to ensure that the identities and rights of the system users are shared without exposing their credentials.

Finally, a simulation shows the secure way of sharing resources distributed amongst the different systems, in order to create a study about the security that the designed system offers.

Índice

1. Introducción 1
1.1. Contexto y justificación del Trabajo 1
1.2. Objetivos del Trabajo 2
1.3. Enfoque y método seguido 3
1.4. Planificación del Trabajo 4
1.5. Estado del arte 7
1.6. Breve sumario de productos obtenidos 9
1.7. Breve descripción de los otros capítulos de la memoria 10
2. Conceptos de seguridad 11
2.1. Autenticación 11
2.1.1. Tipos de autenticación 11
2.1.2. Autenticación doble o múltiple factor 12
2.2. Autorización 13
2.2.1. Control de acceso 13
2.3. Protocolos de comunicación y cifrado de datos 14
2.3.1. Criptografía simétrica 15
2.3.2. Criptografía asimétrica 15
2.3.3. SSL/TLS 16
2.4. Seguridad en web APIs 17
2.4.1. Autenticación basada en cookies de sesión 17
2.4.2. Autenticación basada en *tokens* 18
3. Gestión de accesos e identidades (IAM) 25
3.1. Identidad digital 25
3.1.1. Correspondencia entre Identidad real e identidad digital 25
3.1.2. Elementos y características de la identidad digital 26
3.1.3. Ciclo de vida de la identidad digital 26
3.2. Sistemas de gestión de identidad 28
3.3. *Single Sign-On* (SSO) 29
3.3.1. Ventajas de SSO 30
3.3.2. Inconvenientes de SSO 30
3.3.3. Tipos y estándares SSO 30
3.4. Sistemas de gestión de identidad federada 33
3.4.1. OpenID 34
3.4.2. OAuth 37
4. Prueba de concepto de un sistema de autenticación y autorización 42
4.1. Requerimientos del sistema 42
4.2. Diseño de la arquitectura del sistema 42
5. Conclusiones 48
6. Glosario 49
7. Bibliografía 50
8. Anexos 54
8.1. Estructura de la aplicación cliente 54
8.2. Estructura de la aplicación servidor 55

Lista de figuras

Ilustración 1: Diagrama de Gantt con la planificación temporal de las tareas 6
Ilustración 2: Esquema de pasos para una autenticación 12
Ilustración 3 Estructura de un token JWS (serialización compacta) 20
Ilustración 4 Estructura de un token JWE (serialización compacta) 21
Ilustración 5 Esquema SSO 29
Ilustración 6 Esquema Web-SSO con reverse proxy 32
Ilustración 7 Esquema de autenticación con tickets Kerberos 33
Ilustración 8 Patrón de identidad federada 34
Ilustración 9 Diagrama del protocolo OIDC 35
Ilustración 10 OpenID Connect Authorization Code Flow 36
Ilustración 11 Diagrama de flujo de protocol OAuth 2.0 38
Ilustración 12 Flujo OAuth 2.0 Implicit 40
Ilustración 13 Diagrama de arquitectura 43
Ilustración 14 Diagrama de autenticación con credenciales 44
Ilustración 15 Diagrama de autenticación con un proveedor de identidad 45
Ilustración 16 Diagrama de solicitud de un recurso 46

1. Introducción

1.1. Contexto y justificación del Trabajo

En los últimos años ha habido un gran crecimiento y evolución de la capacidad, seguridad y escalabilidad de los sistemas distribuidos con el que grandes empresas han hecho frente a la enorme demanda de información en Internet.

Hoy en día es habitual que los usuarios compartan información en las redes sociales, consulten el correo electrónico desde la web, realicen cursos online de pago desde el móvil, etc. Cada uno de estos sistemas posee su propio sistema de autenticación y autorización a los recursos, y habitualmente guardan datos sobre la identidad de sus usuarios. Esto puede ocasionar un problema, ya que el usuario debe recordar las credenciales de cada sistema y, además, sus datos personales están almacenados en cada uno de ellos, dificultando su gestión y actualización.

Este problema ha impulsado a grandes empresas como Google o Facebook a ofrecer a la alternativa de autenticarse en sistemas de terceros utilizando un sistema de gestión de identidad federada. De esta forma, la gestión de la identidad y datos personales de los usuarios queda centralizada en un único sistema que no expondrá sus credenciales ni datos personales cuya compartición no esté autorizada.

Mediante una identidad federada, los proveedores de información deben ser capaces de validar que el consumidor es quien dice ser y dispone de acceso a ciertos recursos autorizados mediante una lista de control de acceso (ACL). Para ello, han aparecido varios estándares de autenticación y autorización que facilitarán la transmisión de la identidad del usuario de forma segura.

El presente trabajo final de máster tiene como finalidad el estudio de los estándares existentes para satisfacer la necesidad de compartir información sobre la identidad de los potenciales usuarios de un sistema.

Una vez analizados los diferentes estándares, se diseñará e implementará una solución que permita la autenticación de un cliente y la posterior autorización a diferentes recursos distribuidos utilizando una ACL. Finalmente, se analizarán diferentes aspectos de la seguridad del sistema implementado.

1.2. Objetivos del Trabajo

El principal objetivo de este trabajo es el estudio de los diferentes estándares de autenticación y autorización de sistemas distribuidos y su análisis desde un punto de vista de la seguridad, qué características tienen y sobre qué entornos son aplicables. Para poder analizar al detalle estos estándares, previamente será necesario explicar ciertos conceptos de seguridad y de identidad digital.

Seguidamente, se explicará el concepto de identidad federada y cómo pueden integrarse en un sistema de autenticación. Se detallará que estándar siguen compañías como Google y Facebook para ofrecer la identidad federada a partir de sus credenciales.

Posteriormente, se diseñará e implementará un sistema de autenticación basado en el estándar OAuth 2.0 [6]. Mediante los *tokens* obtenidos durante el proceso de autenticación, se podrá solicitar acceso a ciertos recursos privados.

A modo resumen, los objetivos del TFM serán los siguientes:

Objetivos de estudio y análisis

- Exponer los diferentes conceptos de identidad digital y seguridad involucrados en sistemas de autenticación y autorización en entornos web distribuidos.
- Realizar un análisis detallado de cómo hacer uso de las cookies de sesión y el estándar *JSON Web Token* (JWT) [3] para la autorización de recursos.
- Analizar los estándares de autenticación más destacados hoy en día para sistemas distribuidos.
- Entender el concepto de identidad federada y cómo se integra en sistemas de autenticación.
- Detallar cómo Google y Facebook ofrecen identidad federada a partir de las credenciales.

Objetivos de desarrollo

- Diseñar un sistema de autenticación y autorización basado en el estándar oAuth 2.0 y utilizando *tokens* siguiendo el estándar JWT.
- Definir cómo se puede permitir el acceso a recursos a terceras partes mediante *tokens* JWT.
- Implementar el sistema diseñado simulando un entorno real de Internet.
- Poner a prueba y analizar diferentes aspectos de seguridad del sistema diseñado.

1.3. Enfoque y método seguido

El enfoque del proyecto consiste en desarrollar un sistema de autenticación y autorización para sistemas distribuidos previamente estudiando las soluciones ya existentes y seleccionando la más apropiada para el desarrollo.

El proyecto se divide en dos partes, una más teórica y otra más práctica:

- La primera parte consiste en hacer un análisis de los estándares de autenticación existentes para sistemas distribuidos y profundizar en diversos conceptos de seguridad.

- La segunda parte consiste en seleccionar los estándares más apropiados analizados en la primera parte y diseñar un sistema de autenticación y autorización basándose en estos estándares. Seguidamente, se implementará un prototipo del sistema diseñado.

Para poder conseguir los objetivos, se ha dividido el trabajo en dos entregas: la primera abarcará la parte más teórica y la segunda contendrá el producto desarrollado.

Al analizar el coste en horas necesario para realizar cada una de las partes, la segunda parte destaca sobre la primera. Por este motivo, la estrategia será ir avanzando la parte del desarrollo durante la primera entrega para poder llegar a los objetivos marcados.

Para el desarrollo del proyecto se utilizará una metodología evolutiva / ágil, de manera que se realizará una primera versión sencilla y se iterará en futuras entregas ampliando ciertos apartados a medida que se va desarrollando el proyecto hasta llegar a alcanzar los objetivos y obtener el producto deseado. Se aplicará el estándar definido por la ISO 62304 evolutivo que permite realizar ciclos de desarrollo y analizando los problemas y riesgos en cada ciclo.

De esta manera, durante el desarrollo se estudiará si es necesario redefinir el alcance del proyecto simplificando u omitiendo ciertas partes para poder obtener un mínimo producto viable.

1.4. Planificación del Trabajo

A continuación, se detallan las tareas a realizar divididas en cuatro bloques, uno por entregable, y una planificación temporal de cada una de ellas.

	Tarea	Inicio	Fin	Horas
1	Definición y planificación del TFM			
1.1.	Definir la orientación del trabajo	18/10/2019	18/10/2019	3
1.2.	Contextualizar y justificar el TFM	19/10/2019	19/10/2019	4
1.3.	Definir los objetivos	20/10/2019	22/10/2019	3
1.4.	Definir la metodología a seguir	23/10/2019	24/10/2019	3
1.5.	Listar las tareas a realizar	25/10/2019	27/10/2019	2
1.6.	Planificar las tareas y diagrama de Gantt	27/10/2019	30/10/2019	5
1.7.	Redactar Abstract	30/10/2019	01/10/2019	4
1.7.	**Entrega del plan de trabajo**	**01/10/2019**	**01/10/2019**	**Hito**
2	Análisis e investigación			
2.1.	**Conceptos de Identidad Digital y seguridad**			
2.1.1.	Definir autenticación, autorización y los protocolos	02/10/2019	05/10/2019	6
2.1.2.	Analizar la seguridad en APIs: cookies de sesión, JWT	05/10/2019	08/10/2019	5
2.2.	**Sistemas distribuidos**			
2.2.1.	Analizar los sistemas distribuidos	09/10/2019	12/10/2019	6
2.3.	**Identidad federada**			
2.3.1.	Estudiar los sistemas de gestión de identidad federada	12/10/2019	14/10/2019	4
2.3.2.	Definir del estándar OpenID	14/10/2019	20/10/2019	5
2.3.3.	Definir del estándar OAuth 2.0	21/10/2019	26/10/2019	8
2.3.4.	Analizar la identidad federada en Google y Facebook	27/10/2019	29/10/2019	5
2.4.	**Entrega 2**	**29/10/2019**	**29/10/2019**	**Hito**
3	Implementación de un sistema de autenticación y autorización			
3.1.	**Análisis y diseño del sistema**			
3.1.1.	Detallar los estándares a utilizar	30/10/2019	01/11/2019	3
3.1.2.	Diseñar el sistema y su funcionamiento	01/11/2019	03/11/2019	5
3.2.	**Implementación**			
3.2.1.	Realizar la arquitectura del proyecto	03/11/2019	06/11/2019	8
3.2.2.	Desarrollar el sistema de autenticación	06/11/2019	12/11/2019	14
3.2.3.	Desarrollar el sistema de autorización	12/11/2019	18/11/2019	12

3.2.4.	Desarrollar autenticación con Google y Facebook	18/11/2019	23/11/2019	6
3.3.	**Seguridad**			
3.3.1.	Analizar la seguridad del sistema implementado	23/11/2019	25/11/2019	5
3.3.2.	Proponer puntos de mejora	25/11/2019	26/11/2019	2
3.4.	**Entrega 3**	**26/11/2019**	**26/11/2019**	**Hito**
4	Presentación y defensa del TFM			
4.1.	Redactar las conclusiones del TFM	27/11/2019	29/11/2019	5
4.2.	Redactar y revisar la memoria final	29/11/2019	31/12/2019	20
4.3.	**Entrega 4: Memoria final**	**31/12/2019**	**31/12/2019**	**Hito**
4.4.	Elaboración del vídeo del TFM	01/01/2020	07/01/2020	8
4.4.	**Entrega 5: Presentación en vídeo**	**07/01/2020**	**07/01/2020**	**Hito**
4.5.	**Defensa del TFM**	**13/01/2020**	**17/01/2020**	**Hito**

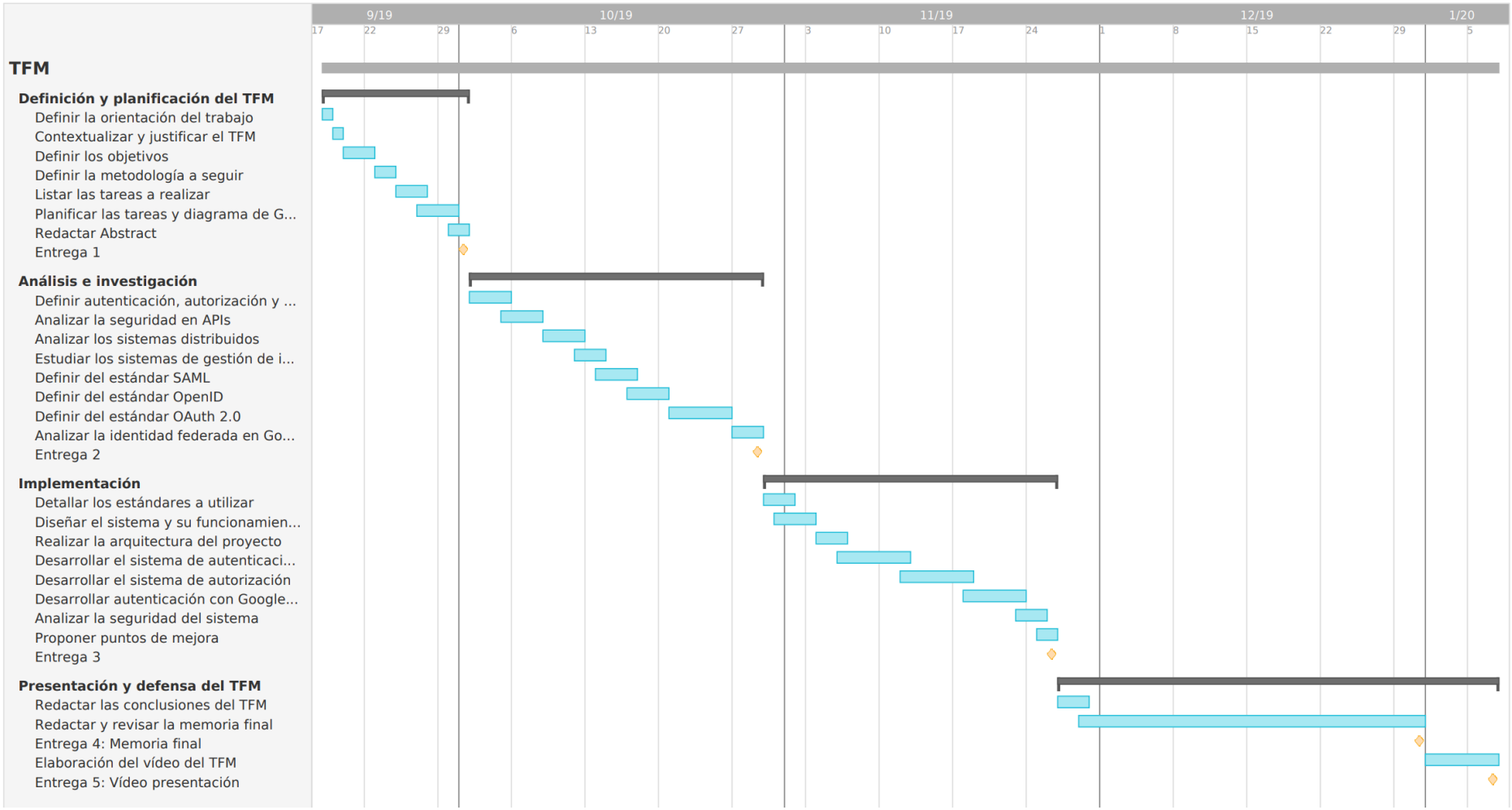

Ilustración 1: Diagrama de Gantt con la planificación temporal de las tareas

1.5. Estado del arte

El primer objetivo del presente trabajo es realizar un estado del arte de diferentes sistemas de autenticación y autorización que puedan ser utilizados en sistemas distribuidos. El segundo objetivo será implementar una prueba de concepto de un sistema de autenticación basado en alguno de los protocolos y estándares más utilizados hoy en día.

En [12] se presenta una solución de un protocolo de autenticación y autorización basado en la nube que proteja de forma segura la información de identidad en este tipo de entornos. Esta solución extiende de estándares ya existentes que son previamente analizados como son:

- *OpenID* [7], un standard para autenticación de usuarios que elimina la necesidad de mantener diferentes identidades con terceras partes y comunica el proveedor de identidad con la parte de confianza.

- *Shibboleth* [11], proporciona un sistema *Single sign on* (SSO) de código abierto que permite autenticarse con identidad federada, evitando la necesidad de requerir diferentes identidades y contraseñas para diferentes servicios.

- *OAuth* [6], estándar de código abierto para delegar autorizaciones, proporcionando una solución que permite a los clientes de cualquier organización acceder a recursos de otros clientes dentro o fuera de la organización. También proporciona autorización a terceros para acceder a servicios y recursos sin compartir las credenciales de usuario. Este sistema es utilizado hoy en día por grandes compañías como Google y Facebook para proporcionar identidad federada.

En [13] se presenta un informe que analiza varios métodos de SSO y sus ventajas al adoptarlos. Seguidamente, debate sobre la implementación de varios tipos de SSO y los protocolos que se utilizan. En el artículo nos clasifica los métodos de SSO según donde se despliegan (Intranet, Extranet o Internet) y cómo se despliegan (con una arquitectura SSO simple o compleja). En una arquitectura compleja, tenemos dos tipos de credenciales: único y múltiples, y según el tipo existen diferentes protocolos.

Los protocolos que se analizan son:

- *Kerberos* [4], que implementa un protocolo de autenticación distribuido basado en tokens.

- *Security Assertion Markup Language* (SAML) [10] estándar de Código abierto basado en XML que permite intercambiar datos de autenticación y autorización entre un proveedor de identidad y un proveedor de servicios.

- *OpenID* [7], citado anteriormente en este apartado.

- *BrowserID* [1], es un sistema de identidad descentralizado que permite iniciar sesión desde cualquier sitio web con una sola contraseña. Implementa el protocolo de correo electrónico verificado creado por Mozilla.

Muchos de los sistemas diseñados para resolver la autenticación y autorización en sistemas distribuidos están basados en estos protocolos analizados anteriormente. Por ejemplo, la *NASA* ha diseñado *UFAA* [9], un framework de autenticación y autorización basado en el estándar *OAuth* 2.0 para la autorización federada y el estándar *JSON Web Token* (JWT) [3] para los tokens de acceso y *JSON Web Signature* (JWS) [2] para firmar los datos y verificar que el mensaje recibido no ha sido modificado. Este framework es la base para intercambiar datos de forma segura y de confianza entre el *Flight Information Management System* (FIMS) y la red *USS* y dentro de la propia red *USS*.

En [5] se presenta una arquitectura para mejorar la seguridad y fiabilidad de las infraestructuras de autenticación y autorización basadas en RADIUS [8] y OpenID [7]. En concreto, se presentan dos prototipos, cada uno basado en uno de los protocolos anteriores, y se evalúa su seguridad y robustez en diferentes entornos. La conclusión a la que se llega es que ambos sistemas son seguros contra diferentes ataques como *DoS*, tolerantes a fallos y están preparados para ser utilizados en entornos empresariales con gran volumen de usuarios donde se comparta información sensible.

1.6. Breve sumario de productos obtenidos

El TFM se divide en entregas parciales que se irán realizando durante todo el semestre. El conjunto de los entregables será el resultado final del trabajo:

- Entrega 1: La primera entrega está formada por el plan de trabajo donde se define el ámbito del trabajo, los objetivos que se quieren alcanzar, la metodología a seguir, un desglose y planificación temporal de las tareas a realizar para alcanzar estos objetivos y un estado del arte.

- Entrega 2: La segunda entrega se trata de una entrega parcial que contiene parte del análisis de conceptos teóricos de seguridad y estado del arte de los estándares de autenticación y autorización existentes. Por otro lado, se presenta el diseño de la prueba de concepto de un sistema de autenticación y se inicia su desarrollo utilizando algunos de los estándares nombrados anteriormente.

- Entrega 3: La tercera entrega se trata de una entrega parcial donde se completa el análisis teórico con una comparativa de los estándares existentes y se finaliza la parte de diseño e implementación de la prueba de concepto del sistema de autenticación y autorización.

- Entrega 4: En esta entrega se juntan las entregas anteriores y se finaliza la memoria del TFM completando ciertos apartados como las conclusiones, glosario y anexo. Se mejoran los aspectos formales de todo el documento y se entrega junto con el desarrollo realizado como producto resultante. Se detalla si se han alcanzado los objetivos propuestos al inicio del TFM.

- Entrega 5: Esta entrega consta de la entrega de un vídeo donde se presenta una síntesis del trabajo realizado mediante diapositivas. Incluye también una demostración del funcionamiento de la prueba de concepto realizada en las entregas parciales 2 y 3.

1.7. Breve descripción de los otros capítulos de la memoria

El trabajo final de master está dividido en dos grandes partes. La primera parte consta de un análisis teórico de protocolos y estándares ya existentes para implementar sistemas de autenticación y autorización en sistemas distribuidos. Los dos primeros capítulos están englobados en esta parte.

En el primer capítulo “Conceptos de seguridad” se definirán conceptos necesarios para poder analizar los protocolos y estándares posteriores. Se definirá que se entiende por autenticación, autorización y que técnicas existen para almacenar y gestionar los privilegios de los usuarios. A continuación, se analizarán conceptos de seguridad en web APIs y se definirán estándares utilizados como JWT y cookies de sesión.

El segundo capítulo, también englobado en la parte de análisis teórico, se explicará el concepto de SSO y de “Identidad federada” y se analizarán los sistemas de gestión de identidad federada. A continuación, se definirán los estándares y protocolos más utilizados para la gestión de la identidad federada, como son: OpenID y OAuth.

La segunda parte del TFM consta del diseño e implementación de una prueba de concepto utilizando conceptos y estándares analizados en los apartados anteriores. En concreto, se diseñará un sistema de autenticación y autorización basado en OAuth que permitirá autenticarse mediante sistemas de terceros. Se simulará una autorización de recursos basada en permisos y, desde una aplicación cliente, el usuario podrá autenticarse en el sistema y, según sus privilegios, verá unos recursos u otros. Finalmente, se realizará un análisis de seguridad del sistema implementado.

En el último apartado de conclusiones se detallarán si se han alcanzado los objetivos propuestos y se analizará si la metodología seguida a lo largo del trabajo ha sido la adecuada. Finalmente se expondrán líneas de trabajo futuro que han quedado pendientes en este trabajo.

2. Conceptos de seguridad

En este apartado se definirán algunos conceptos de seguridad que aparecen en los sistemas de información. Inicialmente se definen los básicos de seguridad como son la autenticación y autorización y, seguidamente, se detallan los protocolos seguros para proteger las comunicaciones.

2.1. Autenticación

Se conoce como autenticación al proceso de confirmación que un remitente es quien dice ser a la hora de intentar acceder a un sistema. En este proceso intervienen dos partes:

- **Remitente**: Se refiere al un usuario, programa o máquina que quiere conectarse a un sistema que requiere identificación.

- **Verificador**: Es la parte que verifica la identidad digital del remitente que quiere conectarse al sistema.

2.1.1. Tipos de autenticación

Una autenticación puede efectuarse por diferentes mecanismos, según qué tipo de información del usuario requiera el sistema para identificarle. Se pueden distinguir según estos tres factores:

- **Autenticación por conocimiento**: el remitente facilita información que solo él conoce, como por ejemplo una clave.

- **Autenticación por algo poseído**: el remitente posee algún objeto que le permite identificar su identidad, como por ejemplo una tarjeta inteligente, una tarjeta de coordenadas, llaves de seguridad USB, etc.

- **Autenticación por característica física**: la persona utiliza algún rasgo físico característico para identificarse, por ejemplo: verificación de huella, de voz, de iris, etc. Es necesario emplear algún dispositivo biométrico que permita verificar estas características físicas.

En el ámbito de este trabajo estudiaremos autenticación por conocimiento, en el que el usuario requiere de clave o *token* para realizar la autenticación. En concreto, se analizarán los sistemas de autenticación más utilizados en sistemas

distribuidos. A alto nivel, el proceso de autenticación consta de los pasos mostrados en el siguiente esquema:

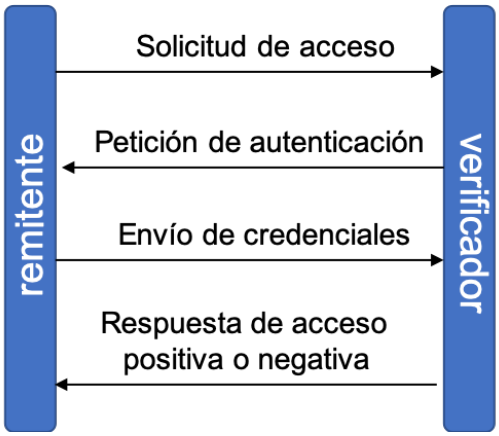

Ilustración 2: Esquema de pasos para una autenticación

2.1.2. Autenticación doble o múltiple factor

La autenticación por conocimiento es la más extendida en la mayoría de sistemas de verificación de identidad. A su vez, es también el mecanismo que presenta más vulnerabilidades, sufriendo ciberataques cada vez más sofisticados que permiten obtener las claves de los usuarios y suplantar su identidad, como por ejemplo *phishing*, malware, etc. Para reducir el éxito estos ataques y garantizar que el usuario es quien dice ser, suele combinarse este mecanismo con algo que posea el usuario.

En los casos que el sistema combina dos tipos de autenticación estamos hablando de verificación en dos pasos. Como se ha comentado anteriormente, el primer factor suele ser una clave conocida por el usuario y el segundo factor un código aleatorio generado por un llavero, una aplicación móvil o una tarjeta inteligente. Un ejemplo sería un usuario que quiere realizar una transacción en un banco y necesita realizar un primer paso se requiere introducir una contraseña (autenticación por conocimiento) y un segundo paso introducir un código aleatorio que ha recibido por SMS en su Smartphone.

La autenticación en dos pasos presenta el siguiente procedimiento:

1. El usuario accede a la pantalla de introducción de credenciales.
2. El usuario inserta el usuario y clave conocidas.
3. El sistema de autenticación valida las credenciales y, en caso de ser correctas, solicita un código aleatorio recibido en su *smartphone* a través de una aplicación, un SMS o una notificación *push*.

4. El usuario inserta este código en la pantalla.
5. El sistema de autenticación verifica el código y responde al usuario indicando si tiene o no acceso al sistema.

2.2. Autorización

Se conoce como autorización al proceso por el cual el sistema autoriza al usuario autenticado a acceder a ciertos recursos protegidos sobre los cuales se le haya concedido autorización previamente. Estos recursos pueden ser ficheros, datos, dispositivos, funciones u otro tipo de información que ofrece el sistema. La autorización debe asegurar la integridad y confidencialidad de los datos, ofreciendo o denegando el acceso de lectura, creación, modificación o borrado. Este acceso debe regirse por el principio de privilegio mínimo, es decir, únicamente se dará acceso a un recurso a aquellos usuarios que los necesiten para realizar su trabajo.

Los sistemas más antiguos eran monousuarios, por lo que no requerían de autorización dado que el usuario autenticado era capaz de acceder a todo lo disponible en el sistema. Poco a poco fueron apareciendo sistemas multiusuario, en los que varios usuarios tienen acceso al sistema y pueden acceder a ciertos recursos si disponen de un rol o permiso que les autorice, de lo contrario se les denegará su acceso. El encargado de realizar esta tarea es el proceso de control de acceso mediante políticas de acceso previamente establecidas.

2.2.1. Control de acceso

Cuando un sistema recibe una petición de acceso a un recurso se activa el proceso de control de acceso que verifica que el consumidor tenga permisos para acceder a dicho recurso. Existen varios mecanismos de autorización que permiten controlar el acceso a los recursos en función de los privilegios del usuario que intenta acceder a ellos. Esto permitirá, por ejemplo, mostrar al usuario ciertas opciones o recursos dependiendo de sus privilegios, evitando que pueda acceder a recursos a los que no está autorizado. A continuación, se muestran algunos de estos mecanismos de autorización:

Discretionary Access Control (DAC)

Este mecanismo, analizado en [19], fue desarrollado para implementar Matrices de Control de Acceso. Las políticas discrecionales definen el control de acceso basado en la identidad de los solicitantes y las reglas de acceso definidas. En

DAC, los usuarios pueden otorgar la autorización a otros usuarios para acceder a los recursos y se utiliza una política administrativa para asignar y conceder los privilegios.

Mandatory Access Control (MAC)

En MAC, analizado en [19], los usuarios no tienen la autoridad para cambiar las reglas de acceso y existe la figura de un administrador de políticas de seguridad que controla la asignación de las políticas MAC de forma centralizada.

Listas de control de acceso (ACL)

Las ACL son listas de permisos asociadas a un recurso (fichero, dato, dispositivo, función, etc.) en función de un usuario o grupo de usuarios. Mediante las ACL, se puede determinar qué usuario o usuarios pueden acceder a un recurso concreto. Pueden configurarse:

- Por usuario.
- Por grupo de usuarios.
- A través de la máscara de permisos.

Un sistema o aplicación puede desarrollar su propia lista de control de acceso o bien, debido a que la mayoría de sistemas operativos disponen de este concepto, se puede utilizar la que ofrece el propio sistema operativo donde corre la aplicación.

Role-based Access control (RBAC)

Analizado en [19], los roles son asignados por el administrador del sistema de forma estática. En RBAC, se controla el acceso a los recursos dependiendo de los roles de los usuarios individuales dentro de una organización. Es compatible con un tipo de control de acceso DAC y MAC. El control de acceso a los recursos es fácil de configurar ya que se limita a definir los roles existentes para acceder a los recursos y se asignan a los usuarios que se desee dar acceso a estos.

2.3. Protocolos de comunicación y cifrado de datos

En los anteriores apartados se ha descrito qué entendemos por autenticación y autorización y sus tipos. Para ambos procesos la información debe viajar desde un cliente a uno o varios servidores los cuales emitirán una respuesta de nuevo al cliente.

Estos datos, a menudo sensibles como pueden ser las credenciales de usuario a la hora de autenticarse, deben viajar de forma segura o quedarán expuestos para que un atacante pueda leerlos, interpretarlos y utilizarlos en su beneficio. Así pues, es muy importante que los datos siempre se transmitan por un canal seguro, cifrado, tanto si viajan por una red abierta a internet como una red interna.

A continuación, se describen protocolos y herramientas que permiten asegurar que los datos viajan de forma segura.

2.3.1. Criptografía simétrica

Una clave simétrica [20] es una cadena de caracteres, similar a una contraseña, que es utilizada para encriptar. Se trata de un método criptográfico en el cual se utiliza la misma clave para cifrar como para descifrar la información compartida entre un emisor y un receptor.

Solo los elementos que conozcan esta clave podrán descifrar la información encriptada con ella. Uno de los problemas de este tipo de claves es a la hora de gestionarlas cuando deben ser compartidas a N elementos, ya que puede quedar comprometida la seguridad del sistema si alguno de los elementos conocedores de la clave la comparte a un elemento no autorizado o ésta es transmitida por un canal no seguro.

La seguridad de los sistemas que utilizan este tipo de claves se basa en la seguridad de la clave y no en el algoritmo utilizado para cifrar los datos. Uno de los algoritmos más utilizados para encriptar con clave simétrica es AES-256, ya que computacionalmente tiene un coste muy bajo.

2.3.2. Criptografía asimétrica

La criptografía asimétrica, también conocida como criptografía de clave pública, es el método criptográfico que utiliza un par de llaves, pública y privada, para cifrar y descifrar un mensaje.
La clave pública puede compartirse sin tener en cuenta la seguridad, ya que disponer la clave pública no permite obtener el contenido del mensaje cifrado.

La clave privada no debe compartirse, solo el propietario debe tener acceso a ella.

El funcionamiento para garantizar la confidencialidad es el siguiente: el emisor emite un mensaje a un destinatario encriptándolo con su clave pública que previamente ha compartido. Cuando el destinatario reciba el mensaje podrá descifrarlo utilizando su clave privada que solo él conoce.

El funcionamiento para garantizar la identificación y autentificación del remitente es el siguiente: el propietario del par de claves usa la clave privada para cifrar un mensaje y este se podrá descifrar utilizando la clave pública. En esta idea se basa el concepto de firma electrónica, donde se desea garantizar que el firmante es el propietario del par de claves.

En este tipo de sistemas se soluciona el problema de compartición de claves de los sistemas que utilizan claves simétricas. En este caso, la clave privada no se comparte y la clave pública puede compartirse sin seguridad.

2.3.3. SSL/TLS

SSL (*Secure Socket Layer*) es el predecesor del protocolo TLS (*Transport Layer Security*) [22] y ambos son protocolos criptográficos que proporcionan privacidad e integridad en la comunicación entre dos puntos en una red. De esta forma se garantiza que la información transmitida no pueda ser interceptada ni modificada por elementos que no estén autorizados. Los únicos elementos que podrán tener acceso a la información serán el emisor y el receptor de la comunicación.

El protocolo SSL se utiliza entre la capa de aplicación y la capa de transporte y normalmente se utiliza junto al protocolo HTTP, convirtiéndose así en la versión segura del protocolo llamada HTTPS. Este protocolo permite transmitir información cifrada y de forma segura en un entorno web, comúnmente Internet.

En estos protocolos se utilizan certificados X.509, es decir criptografía asimétrica (véase 2.3.2.) para autenticar a la contraparte con quien se están comunicando, y para intercambiar una llave simétrica. Con esto se permite garantizar la confidencialidad, la integridad y la autenticación del mensaje.

Fases del protocolo SSL

- Negociar el algoritmo utilizado para la comunicación entre ambas partes, normalmente un cliente y un servidor.

- Intercambio de las claves públicas y autenticación basada en certificados digitales.

- Cifrado del tráfico basado en criptografía simétrica (véase 2.3.1.). El navegador genera una clave simétrica de sesión y la encripta mediante la clave pública que obtuvo del certificado que había mandado el servidor.

2.4. Seguridad en web APIs

Los primeros protocolos y métodos de seguridad de la capa transporte se realizaron para garantizar la seguridad de la interfaz de programación de aplicaciones (API). A continuación, se muestran algunos conceptos para implementar la seguridad en APIs.

2.4.1. Autenticación basada en cookies de sesión

Los primeros sistemas utilizaban las credenciales (usuario y contraseña) para realizar la autenticación y eran transmitidas por un canal seguro SSL/TLS [23]. El proceso era el siguiente:

1. El usuario accede a una página que requiere autenticación.
2. El servidor devuelve el código HTTP 401 *Unauthorized*.
3. El usuario introduce las credenciales en un formulario de autenticación.
4. El navegador concatena el nombre de usuario, el carácter ":" y la contraseña y aplica la función base64 sobre ella, es decir, un algoritmo reversible.
5. Se envía al servidor en una cabecera HTTP llamada *Authorization* y con el prefijo *Basic*. Debe enviarse en un canal seguro HTTPS.
6. El servidor recibe la petición y utiliza la cabecera para validar la identidad del usuario. Revisa los permisos del usuario y, si dispone de permisos, responde con el recurso solicitado. Además, se genera un identificador de sesión y se adjunta como una *cookie* a la respuesta.
7. El navegador muestra el recurso solicitado al usuario. A partir de ahora, todas las peticiones que se realicen incluirán el identificador de sesión como una *cookie*.
8. El servidor valida la cookie de sesión recibida con la petición y, mediante esta, puede recuperar la identidad y permisos del usuario para decidir si tiene acceso a los recursos solicitados.

Este sistema es válido para aplicaciones web, pero no funciona en el caso de *APIs* donde no interviene un navegador web y que también pueden ser consumidas por otros sistemas. Por ejemplo, en el caso de querer importar o exportar datos de un sistema a otro mediante una *API Rest* o la API destinada a ser consumida por aplicaciones móviles nativas.

En este tipo de escenarios no es habitual disponer de un nombre de usuario y contraseña ni son eficaces las cookies, ya que no hay control sobre cuándo se envían debido a que esta era tarea del navegador en aplicaciones web.

2.4.2. Autenticación basada en *tokens*

Este tipo de autenticación es cada vez más utilizada por los nuevos sistemas debido a que es compatible con SPAs, aplicaciones móviles nativas y APIs. Existen diferentes estándares de generación de *tokens* pero uno ha destacado sobre el resto: *JSON Web Tokens*.

JSON Web Tokens (JWT)

JWT [3] es un estándar de código abierto basado en JSON que permite crear *tokens* de acceso propagando en él la identidad y privilegios del consumidor del servicio. El *token* se firma mediante la clave del servidor, de manera que el cliente y el servidor pueden verificar que el *token* es válido y legítimo.

Los JWT están formados por tres partes:

- **Encabezado** (*header*): identifica el algoritmo utilizado para generar la firma, normalmente HS256.
- **Contenido** (*payload*): contiene la información sobre la identidad del usuario y sus privilegios.
- **Firma** (*signature*): se calcula codificando el encabezado y el contenido en base64url, concatenándose ambas partes con un punto.

Pasos de autenticación basada en *tokens* JWT:

1. El consumidor realiza una petición de autenticación, ya sea mediante credenciales o mediante un sistema de identidad federada.
2. El servidor valida los datos y devuelve un JWT que debe ser guardado por el consumidor.
3. El consumidor hace una petición a un recurso protegido y añade una cabecera HTTP *Authorization,* utilizando el esquema *Bearer*, con el *token* obtenido en el paso 2.
4. El proveedor del servicio valida que el *token* JWT exista y sea válido. En caso afirmativo, se obtendrá la identidad y privilegios del usuario contenidos en el token para validar si dispone de acceso al recurso protegido solicitado.
5. En caso de disponer de permisos, el proveedor del servicio enviará el recurso al consumidor.

Este mecanismo de autenticación no dispone de estado, es *stateless*. A diferencia de la autenticación con cookies, el proveedor de identidad y de servicio no necesita almacenar ninguna sesión del consumidor.

Por otro lado, el hecho de que el *token* contenga en su "contenido" la identidad y privilegios del consumidor, reduce los accesos a la base de datos u otras fuentes de información cada vez que lleguen peticiones a recursos protegidos. Esto hace que el sistema sea más rápido, pero, a su vez, puede provocar temporalmente inconsistencia de datos si estos han cambiado posteriormente a la generación del *token*.

Además de la identidad y privilegios del consumidor, el estándar define otros campos que pueden incluirse en los *tokens* JWT. A continuación, se muestran los más relevantes:

- *Expiration time* (exp): Indica la marca temporal a partir de la cual el JWT deja de ser válido.
- *Issued at* (iat): Identifica la marca temporal en qué el JWT fue emitido.
- *JWT ID* (jti): Identificador único del *token* incluso entre diferentes proveedores de servicio.

Los siguientes campos pueden ser utilizados en el encabezado:

- *Token type* (typ): Parámetro opcional que define el formato del contenido. Es recomendado utilizar el valor JWT.
- *Content type* (cty): Usado para definir la información estructural sobre el JWT. Recomendable utilizarlo solo en el caso de firma o cifrado anidado. Su valor debe ser JWT.
- *Message authentication code algorithm* (alg): En este campo se especifica el algoritmo para verificar la firma del *token*.

El estándar JWT es el nombre genérico para dos implementaciones es en función de cómo se presenta el *payload* del *token*.

- **JWS**: el *payload* está codificado y firmado, por lo que puede verificarse la integridad de su contenido.
- **JWE**: el *payload* está encriptado, por lo que su contenido no puede ser leído por otras partes.

A continuación, se detalla cada uno de ellos.

JWS (JSON Web Signature)

En el estándar JWS [2] se firma el contenido mediante una clave secreta que permite al servidor verificar que no ha sido modificado, garantizando así la integridad del contenido. El contenido de un *token* JWS está codificado en Base64 y puede decodificarse para ver los datos que contiene, por lo que es aconsejable que no contenga datos sensibles o confidenciales.

A continuación (Ilustración 3) se muestra la estructura de un *token* JWS [25].

Ilustración 3 Estructura de un token JWS (serialización compacta)

Para firmar un contenido siguiendo el estándar JWS, deben seguirse los siguientes pasos [25]:

1. Crear un objeto JSON incluyendo todos los parámetros del encabezado que expresen las propiedades criptográficas del *token* que se desee obtener. Puede expresarse mediante cualquiera de estos elementos de encabezado: *jku, jwk, kid, x5u, x5c, x5t y x5t#s256*. El resultado se conoce como ***JOSE Header***.
2. Codificar el encabezado obtenido en el paso 1 en *Base64url-encoded*. De esta manera, se produce el primer elemento del *token* JWS.
3. Crear el contenido a firmar, conocido como **JWS *Payload***. Este contenido no debe ser necesariamente un JSON, puede tener otro formato, como por ejemplo XML.
4. Codificar el *payload* obtenido en el paso anterior en *Base64url-encoded*. De esta manera se produce el segundo elemento del *token* JWS.
5. Crear el mensaje para calcular la firma digital o MAC. Se concatena mediante un punto el primer elemento del *token* obtenido dele paso 2 y el segundo elemento obtenido del paso 4. Se realiza la función ASCII sobre la concatenación.

   ASCII(BASE64URL-ENCODE(UTF8(JOSE Header)) '.' BASE64URL-ENCODE(JWS Payload))

6. Calcular la firma sobre el mensaje construido en el paso anterior, conocida como ***JWS Signature***. Se utiliza el algoritmo definido en el parámetro "alg" del encabezado y la clave privada correspondiente a la clave pública que contiene el encabezado.

7. Codificar en Base64url de la firma obtenida en el paso anterior. De esta manera se produce el tercer elemento del *token* JWS.
8. Concatenar con un punto los tres elementos del *token* JWS obtenidos en los pasos anteriores, tal y como se muestra en la Ilustración 3.

Para distinguir si se trata de un *token* JWS o uno JWE (contenido cifrado) [24] se debe mirar el valor del parámetro de encabezado "alg", detallado en el apartado anterior. Se tratará de un *token* JWS si su valor es una firma digital, un algoritmo MAC o el valor "*none*". Otro parámetro de encabezado que nos permite distinguir el tipo de *token* es "enc" (algoritmo de cifrado), que en el caso de un *token* JWS no debe existir.

JWE (JSON Web Encryption)

En el estándar JWE [26] se encripta el contenido evitando que terceras partes puedan acceder a la información. El contenido cifrado se representa mediante el uso de estructuras de datos basadas en JSON pero, al igual que en el estándar JWS, el contenido a encriptar no tiene que ser necesariamente un JSON, puede ser otro formato como por ejemplo XML o texto plano.

Un *token* JWE está formado por 5 elementos separados por un punto. En la Ilustración 4 se muestra la estructura y el orden de los elementos [25].

Ilustración 4 Estructura de un token JWE (serialización compacta)

El primer elemento de un *token* JWE es el encabezado, conocido como ***JOSE Header***. Su estructura contiene los mismos parámetros que en un *token* JWS, con la diferencia que se pueden añadir dos parámetros nuevos:

- *enc*: algoritmo de cifrado del contenido. Debe ser un algoritmo simétrico AEAD (*Authenticated Encryption with Associated Data*), que se trata de una operación de cifrado en bloque que proporciona la confidencialidad, integridad y autenticidad de los datos.
- *zip*: algoritmo de compresión en el caso que se desea añadir compresión al *payload*.

En el caso de un *token* JWE, el parámetro del encabezado "*alg*" define el algoritmo de encriptación para encriptar la CEK (*Content Encryption Key*). Por ejemplo, el siguiente *JOSE Header* utilizará el algoritmo *A256GCM* para

encriptar el contenido y el algoritmo *RSA1_5* para generar la clave encriptada de JWE, el segundo elemento de un *token* JWE, la ***JWE Encrypted Key***:

{ “alg”: “RSA1_5”, “enc”: “A256GCM” }

El tercer elemento de un *token* JWE, el ***JWE Initialization Vector***, se trata de un valor numérico aleatorio y se usará junto a la clave secreta para encriptar los datos. El algoritmo de encriptación utilizará el vector de inicialización durante el proceso de cifrado de los datos. Por otro lado, este vector será necesario para que el destinatario descifre el mensaje, motivo por el cual se debe añadir el parámetro “iv” en el *token*. En el caso que el algoritmo de encriptación no necesite vector de inicialización, su valor será vacío.

El cuarto elemento de un *token* JWE corresponde al *Base64url-encoded* del *payload* encriptado (***Ciphertext***). El contenido se encripta usando la CEK, el vector de inicialización y AAD (*Additional Authentication Data*). El algoritmo de encriptación usado es el definido en el parámetro “enc” del encabezado JWE.

El *Base64url-encoded* del valor del ***JWE Authenticated Tag*** es el quinto elemento de un *token* JWE. Este valor se genera durante el proceso de encriptación AEAD, junto con el texto cifrado. El *JWE Authenticated Tag* asegura la integridad del contenido encriptado y de la AAD.

Para firmar un contenido siguiendo el estándar JWE deben seguirse los siguientes pasos [25]:

1. Utilizar el algoritmo definido en el parámetro “alg” del *JOSE Header* para calcular el modo de administración de claves.
2. Calcular el CEK y la *JWE Encrypted Key* en función del modo de administración de claves obtenido del punto anterior. Este CEK será utilizado para cifrar el contenido en un paso posterior.
3. Codificar en *Basee64url-eencoded* el valor de la ***JWE Encrypted Key*** para obtener el segundo elemento de un *token* JWE.
4. Generar el **vector de inicialización JWE** mediante un valor aleatorio y codificarlo en *Base64url-encoded*. De esta forma se obtiene el tercer elemento de un *token* JWE.
5. Comprimir el contenido en caso de tener definido un algoritmo de compresión en el parámetro “zip” del encabezado.
6. Construir el JSON con todos los parámetros que definen el ***JOSE Header***, incluido el vector de inicialización generado anteriormente. Codificar en Base64url-encoded para obtener el primer elemento de un *token* JWE.
7. Crear el valor ASCII del encabezado codificado en el paso anterior para obtener los AAD.

8. Cifrar el contenido comprimido utilizando el CEK, el vector de inicialización JWE y los AAD. Se utilizará el algoritmo de cifrado definido en el parámetro “enc” del *JOSE Header*.
9. El algoritmo definido en el parámetro “enc” es un algoritmo AEAD. Después del proceso de encriptación producirá el texto cifrado y el *Authentication Tag*.
10. Codificar en *Base64url-encoded* el texto cifrado (***ciphertext***) generado en el paso anterior. El resultado corresponde al cuarto elemento de un *token* JWE.
11. Codificar en *Base64url-encoded* el valor de la ***Authentication Tag***. El resultado corresponde al quinto elemento de un *token* JWE.
12. Concatenar los cinco elementos obtenidos en los pasos anteriores mediante el caracter punto, tal y como se muestra en la Ilustración 4.

Para distinguir si se trata de un *token* JWE en lugar de un token JWS [24] se debe mirar si el parámetro “alg” del encabezado representa un algoritmo de cifrado de clave, ajuste de clave, acuerdo de clave directa, acuerdo de clave con ajuste de clave o cifrado directo. Por otro lado, si existe el parámetro “enc” (algoritmo de cifrado) en el encabezado, también indica que se trata de un *token* JWE.

Seguridad de los *tokens* JWT

Según el tipo de *token* que se genere y sus parámetros, ofrece una seguridad diferente.

En el caso de tener un JWS sin firma, es decir, un *token* que contiene el parámetro “alg” con valor “*none*”, se tratará de un *token* JWT inseguro. En este caso, será el canal de comunicación TLS el que proporcionará seguridad a los datos, ya que asegura su integridad y confidencialidad. Normalmente, un *token* JWT es utilizado como *Authorization Bearer header* en peticiones HTTPS, como por ejemplo en el caso de Oauth 2.0.

En el caso de generar un *token* JWS con firma, se protegerá la integridad del contenido, por lo que:

- Con un ataque *Man-in-the-middle* se podrá ver el contenido, ya que va codificado en Base64 y puede decodificarse.
- Con un ataque *Man-in-the-middle* no se podrá modificar el contenido, ya que el *token* se verificará mediante la firma y esta fallaría en caso de modificación del contenido.

En el caso de generar un *token* JWE, el contenido irá cifrado y se protegerá también su integridad, por lo que:

- Con un ataque Man-in-the-middle no se podrán ver el contenido, ya que va cifrado.
- Con un ataque Man-in-the-middle no se podrá modificar el contenido, ya que la verificación del *token* fallaría.

3. Gestión de accesos e identidades (IAM)

En este apartado se describirán los diferentes tipos de sistemas de gestión de identidad y, en más profundidad, los sistemas de gestión de identidad federada.

Antes de analizar los sistemas de gestión de identidad digital se definirá el concepto de identidad digital, necesario para describir el resto de apartados.

3.1. Identidad digital

La identidad se define como el conjunto de características mediante las cuales se reconoce una cosa, que la hace única respecto a otras. En el caso concreto de la identidad de una persona, se refiere a las características que identifican a la persona como, por ejemplo: nombre, físico, los datos contenidos en el documento de identidad, etc. La identidad no solo hace referencia a personas, también hace referencia a organizaciones, entidades u otras cosas que se identifiquen como diferentes al resto.

La identidad digital [27, 28, 32] esta vinculada a una persona, organización o dispositivo y hace referencia al conjunto de información presente en Internet y las acciones realizadas en este entorno como, por ejemplo: comentarios, opiniones, navegación, aficiones, noticias o publicaciones que hacen referencia a la entidad que se identifica. Se trata pues del conjunto de información sobre una persona o una organización expuesta en Internet y que conforma su imagen en el mundo digital.

Las trazas que dejan los usuarios al navegar, hacer una acción o acceder a un servicio en Internet, están vinculadas a la identidad digital de estos. Debido a que estas trazas pueden contener información confidencial del usuario, cobra mucha importancia la privacidad de los usuarios y la seguridad de los sistemas.

3.1.1. Correspondencia entre Identidad real e identidad digital

La identidad digital no tiene por qué corresponderse con una identidad real de una persona u organización. Sin embargo, las acciones que se hacen bajo la identidad digital tienen consecuencias en el mundo real y viceversa, ya que afecta a su reputación y a la imagen que los demás usuarios construyen sobre ella.

La identidad digital será equivalente a la real cuando se valide y demuestre que un usuario es quien dice ser. Será requerido algún tipo de autenticación para

demostrar que se trata de una persona y pueda, de esta manera, realizar conexiones o transacciones en Internet.

3.1.2. Elementos y características de la identidad digital

La identidad digital está formada por diferentes elementos o atributos de datos, como podrían ser:

- Actividad de búsquedas
- Transacciones electrónicas
- Nombre de usuario
- Nick y avatar
- Publicaciones
- Historial de compras
- Fotografías

Estos elementos pueden contener información sensible del usuario, por lo que se requieren de sistemas que sean capaces de preservar la privacidad de estos. De esta forma, la identidad digital se vincula a uno o más identificadores digitales (correo electrónico, nombre de dominio, etc.) junto con una contraseña que le identifique.

En los últimos años, han crecido los ciberataques que intentan obtener esta información para acceder a recursos protegidos. Por este motivo, es habitual que se vincule la identidad con algún atributo físico como podría ser la huella dactilar o el reconocimiento facial. En algunos sistemas o dispositivos, será necesario validar las credenciales de usuario junto con alguno de estos atributos para poder acceder a ellos.

Un ejemplo de ello podría ser la nueva normativa de pago online conocida como PSD2 (*Payment Service Directive*), que en España entró en vigor el 14 de septiembre de 2019. Esta normativa exige la autenticación en dos pasos para acceder a la banca digital, por lo que se requerirá una contraseña y una clave recibida en un dispositivo móvil del usuario o bien su huella dactilar. De esta forma, el sistema puede vincular la identidad digital con una identidad real de una persona que desee realizar una transacción bancaria.

3.1.3. Ciclo de vida de la identidad digital

Una identidad digital [29] pasa por tres pasos básicos en su ciclo de vida: la creación, el uso y la baja. A continuación, se detalla cada uno de los pasos.

Alta en el sistema

Paso previo a que se pueda utilizar el servicio o acceder al sistema. Normalmente, el alta se hace de forma no presencial, asignando o escogiendo unas credenciales únicas para acceder al sistema. En el caso de sistemas o servicios con riesgos potenciales se puede requerir un alta presencial.

Procedimiento de autenticación

Un usuario podrá autenticarse en el servicio una vez el usuario está dado de alta. Este usuario pasará a estar dado de alta en una sesión activa. En el caso de entornos web, esta sesión se implementa mediante *cookies*, donde se almacena información variada y un identificador de sesión que será utilizado para validarla en el servidor, ya que este será capaz de asociar un cliente a una determinada sesión.

Un sistema puede utilizar uno de estos tipos de autenticación:

- Autenticación con contraseña: el sistema almacena las contraseñas de los usuarios del sistema en un sitio seguro, como por ejemplo en una base de datos que requiere autenticación y que solo el sistema puede acceder. Normalmente se guarda la contraseña encriptada, de manera que se encripta la contraseña recibida y se compara con la almacenada en el sistema.

- Autenticación con certificados electrónicos: Permite al sistema identificar un cliente mediante su certificado de cliente. Como hemos visto anteriormente (véase 2.3. Protocolos de comunicación y cifrado de datos), los protocolos y especificaciones SSL/TLS protegen la capa de transporte i proporcionan confidencialidad, autenticidad e integridad de la información.
  Cuando un cliente se conecta a un sitio web seguro, se crea un canal seguro de comunicación. El servidor envía su certificado que incluye una referencia al emisor, de manera que el cliente podrá decidir si confía o no en el certificado.

- Autenticación *single sign-on*: Este tipo de sistemas permiten acceder a varios servicios a partir de una única autenticación inicial. Se utiliza en entornos distribuidos y con varios proveedores de servicios, por lo que más adelante se estudiará con profundidad, ya que forma parte del ámbito del proyecto.

Baja en el sistema

El sistema debe permitir dar de baja una identidad digital cuando ya no pueda o no se quiera utilizar en este sistema. El procedimiento de baja puede ser un proceso manual gestionado por un administrador o un proceso automático si se trata de sistemas con una gran cantidad de usuarios.

3.2. Sistemas de gestión de identidad

La gran cantidad de servicios disponibles ha provocado que los usuarios dispongan de más de una identidad digital como, por ejemplo: el perfil de usuario en una red social, acceso al correo electrónico, usuario de un foro, etc.

Los proveedores de servicios tendrán que realizar una gestión de estas identidades y, sobretodo, cuando estos servicios se comunican entre sí. Para una correcta gestión de la identidad y comunicación entre servicios, los proveedores de servicios deberán garantizar los siguientes principios básicos de seguridad: confidencialidad, disponibilidad e integridad.

Para poder garantizar estos principios básicos, se requerirá que los sistemas dispongan de servicios de identificación, autenticación y autorización. De esta forma, si un usuario u organización intenta acceder a cierto recurso protegido, el sistema realizará las siguientes comprobaciones:

- Verificación de la identidad: El sistema necesita averiguar quién intenta acceder, es decir, conocer la identidad del usuario. A continuación, se verificará utilizando algún método de autenticación como, por ejemplo, mediante la combinación de un usuario y contraseña. En algunos casos se utilizará una autenticación con múltiple factor para una mayor seguridad.

- Comprobar el nivel de autorización: En este paso, el sistema obtiene las reglas de autorización del usuario identificado en el paso anterior. Mediante un proceso de control de acceso se validará si este usuario tiene o no acceso al recurso protegido solicitado. De esta forma, se asegura la confidencialidad e integridad de los datos almacenados en el sistema.

Un sistema que no interactúa con otros sistemas puede tener su propia implementación de la autenticación y autorización a sus recursos, pero hoy en día no es el caso más habitual. Normalmente, se requiere que los proveedores de servicios interactúen entre ellos y se combinen entre si, por lo que la colaboración entre ellos será de gran importancia.

A raíz de esta necesidad, y debido a la complejidad añadida de la gestión de identidades cuando se combinan diferentes servicios, nacen diferentes soluciones que permiten al usuario acceder a diferentes servicios y recursos con una única identidad digital. En los siguientes apartados se describirán estas soluciones de autenticación y autorización en entornos distribuidos.

3.3. *Single Sign-On* (SSO)

El "Inicio de Sesión Único" o SSO se trata de un procedimiento de autenticación que requiere que el usuario se autentique una sola vez y le habilite para acceder a varios sistemas relacionados entre ellos con una única instancia de identificación.

En la siguiente ilustración (Ilustración 5) se observa a alto nivel como un usuario accede a diferentes servicios con un único acceso [30].

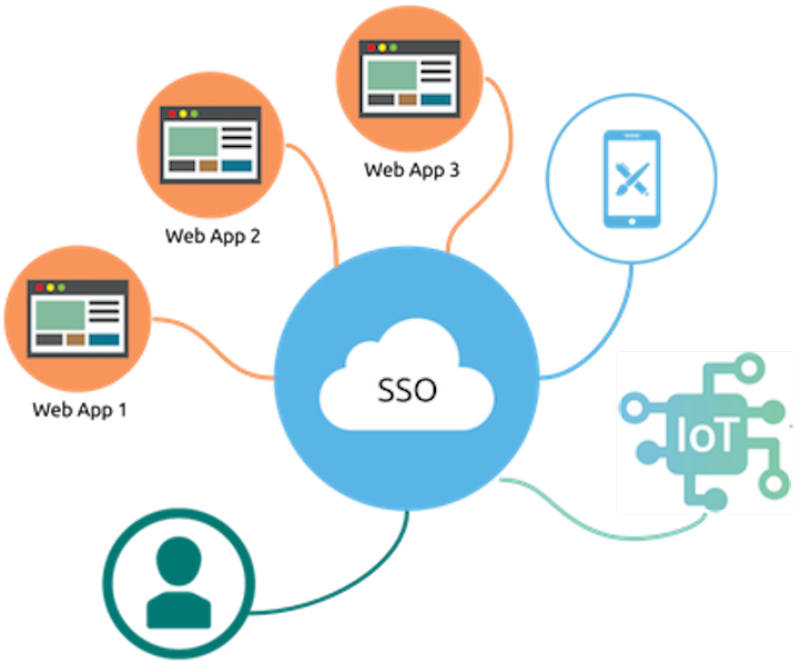

Ilustración 5 Esquema SSO

Como se observa en la ilustración, el usuario introduce sus credenciales en el sistema SSO y, una vez autenticado, puede acceder al resto de páginas web, aplicaciones o servicios. Las diferentes aplicaciones y recursos admiten diferentes mecanismos de autenticación, por lo que el sistema SSO debe almacenar internamente las credenciales utilizadas para la autenticación inicial y traducirlas a las credenciales necesarias para cada una de las aplicaciones.

3.3.1. Ventajas de SSO

A continuación, se enumeran las diferentes ventajas de centralizar la autenticación en un único punto [31]:

- Mitigar el riesgo de acceso a sitios de terceros, ya que las contraseñas se administran y almacenan en un único sitio y no externamente.

- Reduce lo que se conoce como “fatiga de la contraseña”. El usuario no debe recordar las diferentes combinaciones de usuario y contraseña de cada sistema.

- Reducir el tiempo dedicado a volver a introducir las contraseñas en cada uno de los sistemas para una misma identidad.

- Reducción del coste de mantenimiento global del sistema. Por ejemplo, se recibirá un menor número de llamadas de soporte sobre contraseñas y, al añadir un nuevo servicio al sistema, se puede simplificar la parte de autenticación.

3.3.2. Inconvenientes de SSO

A continuación, se enumeran los inconvenientes del inicio de sesión único [31]:

- En caso de robo de credenciales de un usuario, se dispone acceso inmediato a todos los servicios sin pasar ningún mecanismo de seguridad más. Para mitigar este problema, se deberá implementar una política de generación de contraseñas más estricta y segura. También puede combinarse con métodos de autenticación fuertes como tarjetas inteligentes y *tokens* de contraseña de un solo uso.

- La perdida de disponibilidad de SSO puede provocar la denegación de acceso a todos los sistemas unificados bajo él. Este inconveniente hace que la solución SSO no pueda aplicarse en todos los casos, ya que puede haber sistemas en los que se debe garantizar el acceso en todo momento y no sería viable.

3.3.3. Tipos y estándares SSO

A continuación, se detallan los principales tipos de SSO e implementaciones comunes [31].

Enterprise SSO (E-SSO)

También es conocido como "*Legacy SSO*". Funciona para una autenticación primaria, interceptando los requisitos de autenticación de las diferentes aplicaciones del sistema y completándolas con el usuario y contraseña adecuado.

Los sistemas implementados con E-SSO solo permiten interactuar con otros sistemas y aplicaciones que permitan deshabilitar la pantalla de autenticación y que pueda ser completada sin mostrarse. Cuando una TPA (*Third party application*) detecte que requiere autenticación al acceder a ella o cuando la información de autenticación que reciba no sea válida, redirigirá a la pantalla de autenticación genérica del sistema.

Web-SSO

También conocido como "gestión de acceso web" (WAM). Los servicios y aplicaciones de un sistema que utiliza Web-SSO solo pueden ser accedidos vía web.

En este tipo de sistemas, se interceptan los accesos a las aplicaciones o servidores web mediante un *reverse proxy* o un componente instalado en esas aplicaciones [33]. Para poder acceder a una aplicación web será necesario disponer de un *token* de autenticación obtenido en el servicio de autenticación una vez el acceso sea verificado.

En el caso que el sistema requiera un componente instalado en la aplicación, la autenticación y el control de acceso quedará delegada a este agente. El inconveniente de esta solución es que las aplicaciones deben permitir instalar el componente y deben modificar su comportamiento. Dependiendo de la tecnología o sistema operativo esto no siempre es posible.

En el caso de utilizar un *reverse proxy*, este actuará como punto de comunicación entre el navegador web y el servidor web, llevando a cabo operaciones de inicio de sesión y control de acceso. El *proxy* enmascara el servidor exponiendo URLs externas diferentes a las internas, por lo que la aplicación no podrá contener enlaces absolutos a otras aplicaciones internas también accesibles desde un *proxy*. Las ventajas de este sistema son:

- Normalmente no se requiere modificar las aplicaciones protegidas. No depende de la tecnología en la que estén desarrolladas.

- Un mismo servidor web puede ser protegido por varios *reverse proxy*, de manera que se puede proporcionar alta disponibilidad y balanceo de carga.

El esquema [34] del sistema sería como el mostrado en la

Ilustración 6.

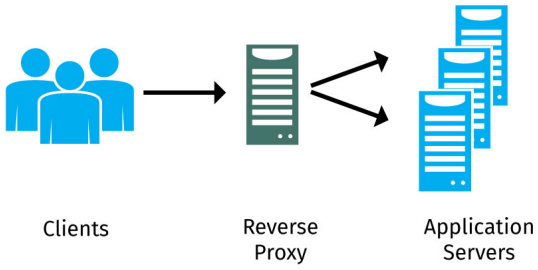

Ilustración 6 Esquema Web-SSO con reverse proxy

Kerberos

Kerberos [35] es un protocolo de autenticación creado por el MIT que utiliza una criptografía de claves simétricas para validar los usuarios con los servicios. Las contraseñas no se envían a través de la red, evitando de esta manera transmitir información de autenticación en canales que pueden ser inseguros.

El protocolo Kerberos ofrece autenticación y gestión de acceso de recursos que se deben proteger dentro del sistema. Se trata de un sistema *single sign-on* y de distribución de claves.

El funcionamiento es el siguiente:

1. El usuario debe iniciar sesión mediante sus credenciales.
2. Si la autenticación es correcta, recibe un Kerberos TGT (*ticket-granting ticket*).
3. Las aplicaciones que requieran autenticación usan el TGT para adquirir tickets de servicio ST.
4. Las aplicaciones pueden demostrar la identidad del usuario al servidor correspondiente sin necesidad de introducir las credenciales de nuevo.

Este modelo esta diseñado como un modelo de cliente-servidor en el que se provee de autenticación en ambas partes, de manera que tanto ambas entidades pueden verificar la identidad uno del otro. Kerberos mantiene una base de datos de claves secretas (KDC) y cada entidad (cliente o servidor) comparte la clave secreta únicamente conocida por ella y Kerberos, que servirá para demostrar la identidad de la entidad.

A continuación (Ilustración 7), se muestra un esquema de funcionamiento del sistema de autenticación con tickets Kerberos [36].

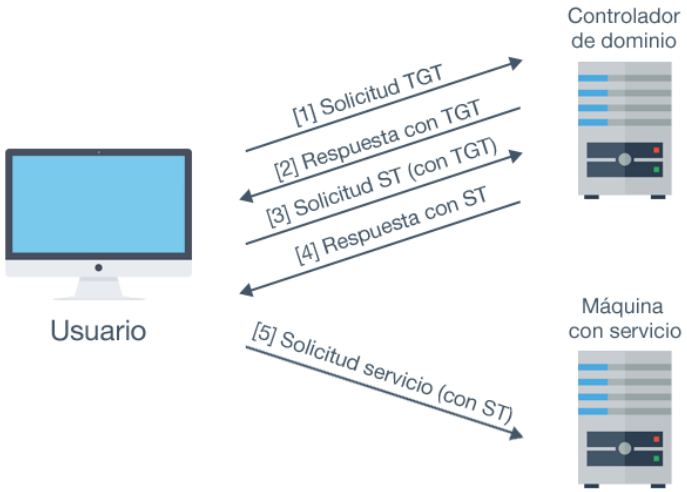

Ilustración 7 Esquema de autenticación con tickets Kerberos

Las principales desventajas de este sistema son las siguientes:

- Si un usuario a parte del administrador tiene acceso a la máquina que emite tickets para autenticaciones, el sistema queda expuesto.
- Es necesario modificar el código de la aplicación para usar Kerberos, ya que se requieren una serie de librerías. Puede haber tecnologías incompatibles.
- Todas las aplicaciones del sistema deben usar Kerberos, ya que otras maneras de autenticación en el sistema pueden ser interceptadas e invalidadas.

3.4. Sistemas de gestión de identidad federada

La identidad federada es una gestión de identidad interdependiente entre organizaciones. Esta relacionada con el SSO, en el que se obtiene un *token* de seguridad (STS) vinculado a un usuario y permite acceder a múltiples sistemas u organizaciones. Proporciona una solución al problema de confiar en identidades de dominios diferentes.

En este tipo de sistemas, se delega la autenticación a un proveedor de identidades de confianza (IdP), simplificando el desarrollo de nuevos sistemas que requieran de autenticación. El IdP emite *tokens* de seguridad que contienen

información sobre el usuario autenticado como, por ejemplo: su identidad, roles, derechos de acceso, etc.
La seguridad del sistema aumenta debido a que no es necesario que el usuario cree credenciales en cada uno de los sistemas, ya que con una única identidad federada se obtiene acceso a todos ellos. Además, los sistemas no tienen acceso a las credenciales originales utilizadas para autenticarse en el proveedor de identidades, de manera que solo obtienen la información de la identidad contenida en el *token*.

En la siguiente ilustración (Ilustración 8) se muestra el patrón de identidad federada cuando un consumidor necesita acceder a un servicio que requiere autenticación [37].

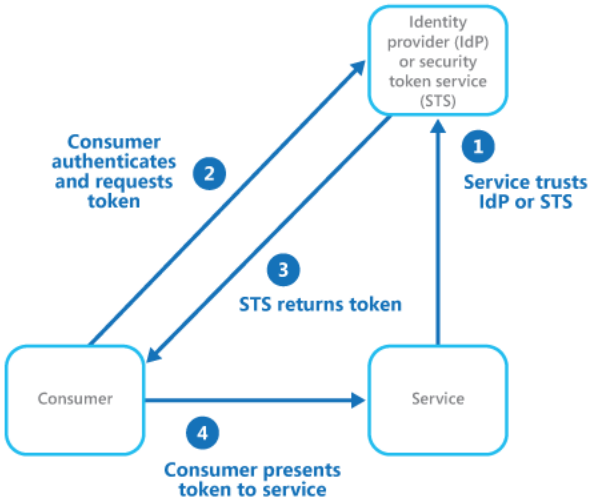

Ilustración 8 Patrón de identidad federada

A continuación, se realiza un análisis de los estándares más utilizados en la actualidad para cubrir la necesidad de la gestión de identidades federadas.

3.4.1. OpenID

OpenID es un estándar de identificación digital descentralizado. Un usuario, proporcionando un identificador XRI, puede ser verificado por cualquier servidor que soporte el protocolo. El estándar no especifica el mecanismo de autenticación, por lo que la seguridad de la conexión depende de la confianza que tenga el cliente OpenID en el proveedor de identidad.

En el año 2007 apareció la última versión de OpenID, que fue la 2.0. En el año 2014 apareció **OpenID Connect** (OIDC) como sustituto de la versión 2.0. En este trabajo analizaremos la versión más actual: OIDC.

OIDC [38] realiza las mismas tareas que OpenID 2.0 pero de forma más amigable, mediante una *API HTTP RESTful*, y se puede utilizar en aplicaciones móviles, aplicaciones web, clientes Javascript, etc. Integra las capacidades de OAuth 2.0, ya que en definitiva no es más que una capa de identidad simple sobre el protocolo OAuth 2.0. El protocolo OAuth 2.0 podía ser utilizado en versiones anteriores de OpenID mediante una extensión, pero en OIDC lo integra dentro del mismo protocolo. Además, OIDC define mecanismos opcionales para la firma robusta y el cifrado.

OpenID Connect Protocol Suite

En el siguiente diagrama (Ilustración 9) se muestran las especificaciones sobre las que se basa el protocolo OpenID Connect [38].

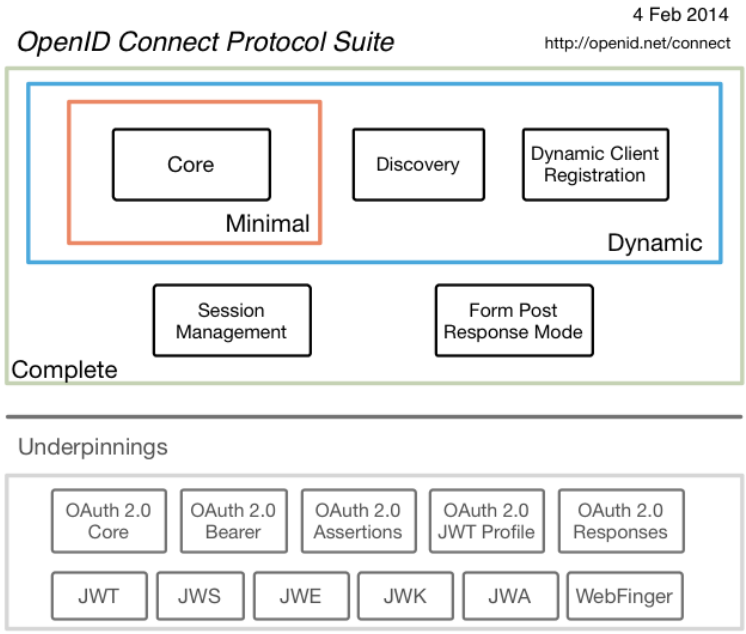

Ilustración 9 Diagrama del protocolo OIDC

En el diagrama se presentan los siguientes elementos:

- ***Core***: define la funcionalidad principal de OpenID Connect. Es una autenticación construida sobre OAuth 2.0. Requerida en cualquier arquitectura que utilice OIDC.
- ***Discovery***: Es opcional y define cómo los clientes descubren dinámicamente información sobre proveedores OpenID.
- ***Dynamic Client Registration***: Es opcional y define cómo los clientes se registran dinámicamente con los proveedores de OpenID.

- ***Session Management***: Es opcional y define como administrar las sesiones de OpenID Connect, incluido la funcionalidad de cierre de sesión.
- ***Form Post Response Mode:*** Es opcional y define cómo devolver los parámetros de respuesta de OAuth 2.0. Utiliza valores en formularios HTML que son enviados automáticamente por el *User Agent* mediante una petición HTTP Post.

Flujo de alto nivel

En el siguiente diagrama (Ilustración 10) se muestra a alto nivel el flujo del código de autorización en OpenID Connect [39]. A continuación, se detallan los pasos.

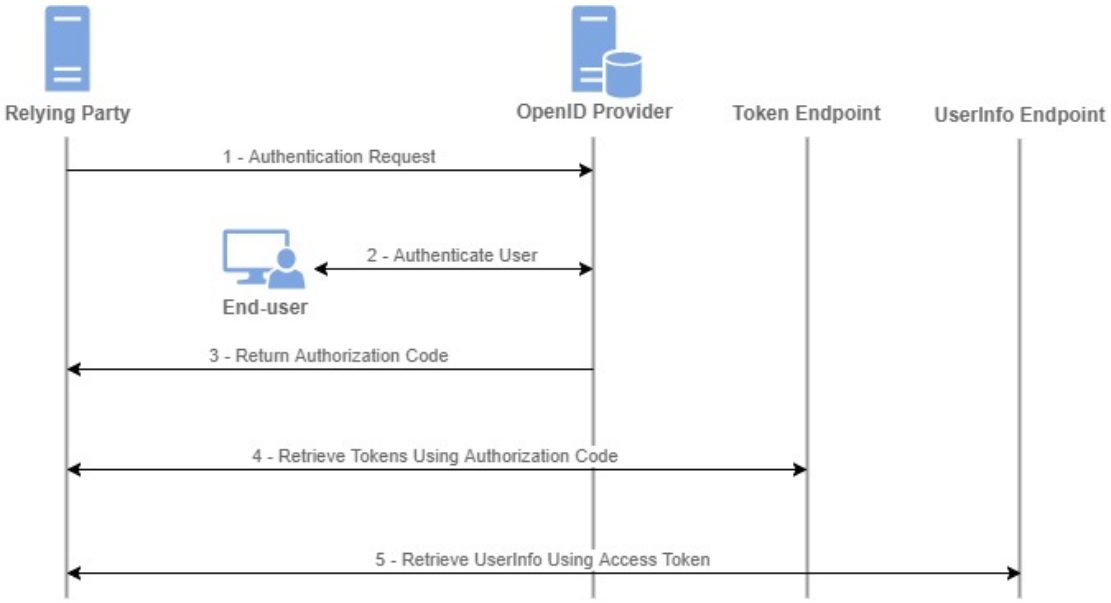

Ilustración 10 OpenID Connect Authorization Code Flow

1- El proveedor de servicios (*Relying Party*) envía una solicitud al proveedor de OpenID para autenticar al usuario final.
2- El proveedor OpenID autentica al usuario final utilizando alguno de los métodos disponibles y obtiene la autorización del usuario final.
3- Una vez el usuario final esté autenticado y se haya autorizado la *request*, el proveedor de OpenID devuelve el código de autorización al proveedor de servicios.
4- El proveedor de servicios contacta con el *Token Endpoint* e intercambia el código de autorización por un *token* de identificación que identifica al usuario final.

5- El proveedor de servicios puede solicitar, de forma opcional, información adicional del usuario, como por ejemplo el correo electrónico. Utilizará el *token* de identificación obtenido en el paso 4.

3.4.2. OAuth

Open Authorization (OAuth) es un estándar abierto de autorización que permite obtener acceso limitado a los sitios web o aplicaciones móviles o de escritorio. Este estándar permite compartir información en el proveedor de servicio con el consumidor sin compartir toda su identidad y protegiendo las credenciales del usuario final.

La primera versión de OAuth apareció en noviembre de 2006 y, la versión más actual es la OAuth 2.0. que apareció en octubre de 2012. OAuth 2.0 no es compatible con OAuth 1.0, y proporciona flujos de autorización específicos para aplicaciones web, de escritorio, nativas y dispositivos. El estándar que se analiza en este trabajo es el OAuth 2.0.

Roles de OAuth

En el estándar OAuth se definen cuatro roles:

- **Propietario de los recursos (*Resource Owner*):** hace referencia al usuario que da la autorización a una aplicación para acceder a su recurso protegido.

- **Servidor de Recursos (*Resource Server*):** almacena los recursos protegidos. Es la API a la que se quiere acceder.

- **Servidor de Autorización (*Authorization Server*):** verifica la identidad del usuario y genera *tokens* de acceso para la aplicación. En este caso, Auth0.

- **Cliente (*Client*):** es la aplicación que desea acceder al recurso protegido. Requiere de autorización previa por parte del usuario.

Registro de la aplicación

Antes que una aplicación pueda utilizar OAuth con un servicio, debe registrarse. Normalmente, el sitio web del servicio dispone de un formulario de registro para proporcionar la información de la aplicación, como por ejemplo:

- Nombre de la aplicación
- Descripción de la aplicación
- Sitio web de la aplicación
- *Redirect URI*: es la URI donde el servicio redireccionará una vez se termine el proceso de autorización. Si ha sido satisfactoria, se recibirá el *token* de acceso en está URI.

Flujo de protocolo

En el siguiente diagrama (Ilustración 11) [40] se muestra cómo interactúan los diferentes roles durante el proceso de autenticación.

Ilustración 11 Diagrama de flujo de protocol OAuth 2.0

A continuación, se detalla cada uno de los pasos:

1- La aplicación (cliente) solicita autorización al propietario de los recursos para acceder a los recursos.
2- La aplicación recibe la autorización si el usuario autoriza la solicitud.
3- La aplicación solicita al servidor de autorizaciones un *Access Token* enviando la autenticación de su propia identidad y la autorización otorgada por el usuario en el paso 2.
4- Si la autorización es válida y la aplicación es autenticada, el servidor de autorización emite un *Access Token* a la aplicación. En este punto finaliza el proceso de autorización.
5- La aplicación solicita un recurso al servidor de recursos y se autentica mediante el *Access Token* obtenido en el paso 4.
6- Si el *Access Token* obtenido es válido, el servidor de recursos provee el recurso a la aplicación.

Authorization grant types

El estándar OAuth 2.0 define cuatro flujos para obtener un *Access Token*, y cada uno de estos flujos se conoce como "*grant types*". Todos ellos están basados en el flujo de protocolo del apartado anterior.

- ***Authorization Code*:** Utilizado en aplicaciones web ejecutadas en el lado del servidor.

- ***Implicit*:** Utilizado en aplicaciones SPA ejecutadas en el navegador del usuario.

- ***Resource Owner Password Credentials*:** utilizado para aplicaciones de confianza, como por ejemplo las que pertenecen al servicio.

- ***Client Credentials*:** utilizado por máquinas que comunican con máquinas, no interviene un navegador.

Cada uno de estos flujos tiene variantes y particularidades, ya que se han optimizado para cada uno de los casos detallados. En el ámbito del proyecto, debido a que se desarrollará un prototipo de una SPA que utilizará identidad federada para acceder a recursos, es necesario ver en detalle el flujo Implícito.

Flujo *Implicit*

Este flujo será el que utilizarán aplicaciones móviles y aplicaciones web SPA, donde no se garantiza la confidencialidad del secreto del cliente ya que es un cliente público. En este tipo de flujo, el *Access Token* se acaba reenviando a la aplicación, por lo que puede estar expuesto a otras aplicaciones o al propio usuario.

En este tipo de flujo no se utilizan *Refresh tokens*, que son utilizados para solicitar un nuevo *Access Token* sin necesidad de volver a solicitar autorización al usuario. Así pues, en este tipo de flujo se podrá acceder al recurso durante un tiempo limitado que corresponde al periodo de validez del *Access Token*.

A continuación (Ilustración 12), se muestra el diagrama del flujo *Implicit* [41].

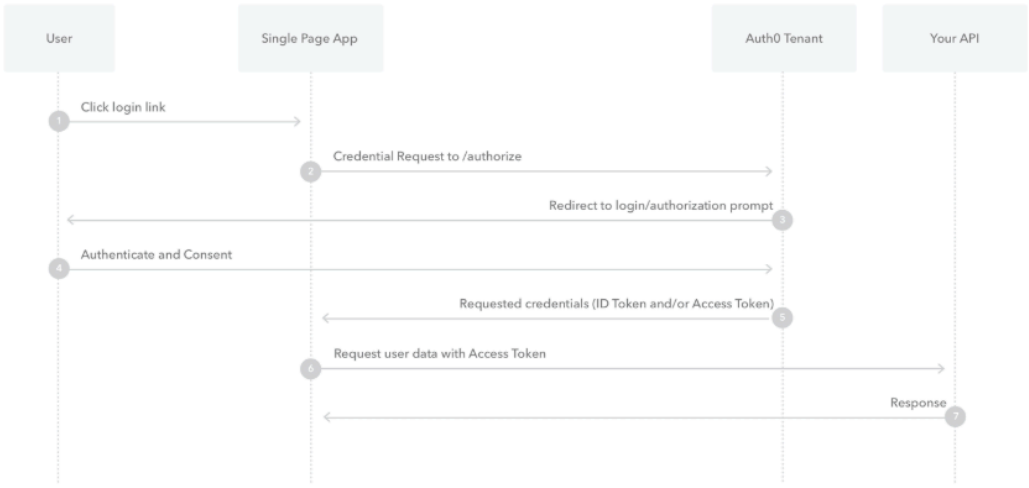

Ilustración 12 Flujo OAuth 2.0 Implicit

Los pasos que se realizan para este tipo de flujo son los siguientes:

1- El usuario realiza un clic para hacer autenticarse en una SPA.
2- El SDK de Auth0 redirige al usuario al servidor de autorización Auth0 pasando el parámetro "*response_type*" que indica el tipo de credenciales solicitadas.
3- El servidor de autorización redirige al usuario a la solicitud de inicio de sesión y autorización.
4- El usuario se autentica y ve una página de consentimiento que enumera los permisos que Auth0 otorgará a la SPA.
5- El servidor de autorización Auth0 dirige de nuevo al usuario al SPA y, dependiendo del parámetro "*response_type*", recibirá:
    - Un *ID Token*: Es un *token* JWT que contiene información del usuario autenticado. Fue introducido por OIDC.
    - Un *Access Token*: JWT que contiene quién ha autorizado y qué permisos y para qué aplicación. Sustituye a las credenciales del usuario.
    - Un *ID Token* y un *Access Token*
6- La SPA podrá utilizar el *Access Token* para llamar a la API.
7- La API responde con los datos solicitados.

Implementación en grandes compañías

Este mecanismo es utilizado por grandes compañías para permitir a los usuarios compartir información limitada sobre sus cuentas con TPA. A continuación, se detalla el estándar soportado por cada una de las compañías en el momento de la realización del trabajo:

- **Facebook**: su API GraphQL solo admite OAuth 2.0.

- **Google**: admite OAuth 2.0 como mecanismo de autenticación recomendado para todas sus APIs.

- **Microsoft**: admite OAuth 2.0 para varias de sus APIs y en su servicio *Azure Active Directory*.

- **Github**: sólo admite OAuth 2.0 como mecanismo de autenticación de sus APIs.

- **Twitter**: admite la versión OAuth 1.0a y OAuth 2.0 *Bearer Token*.

4. Prueba de concepto de un sistema de autenticación y autorización

Una vez analizados los diferentes estándares y protocolos más extendidos en la actualidad, en este apartado se va a diseñar e implementar una prueba de concepto de un sistema de autenticación y autorización.

4.1. Requerimientos del sistema

El sistema estará compuesto por un módulo de autenticación y otro de autorización. El módulo de autenticación debe permitir que el usuario se autentique mediante credenciales o mediante la identidad federada proporcionada por Google o Twitter.

Una vez autenticado, el sistema debe proporcionar una sesión válida para el usuario y un *token* de autenticación mediante el cual pueda realizar peticiones sin necesidad de enviar sus credenciales de nuevo. El sistema debe ser capaz de validar el *token* recibido para verificar que el usuario es quien dice ser. Si la sesión del usuario ha caducado, el sistema debe informar a la aplicación cliente para que el usuario será redirigido a la página de autenticación.

Para un usuario autenticado, el sistema de autorización debe ser capaz de decidir si puede o no acceder al recurso solicitado. Para ello, debe consultar los privilegios del usuario autenticado y ver si tiene los permisos necesarios para acceder al recurso. Los privilegios se obtendrán del *payload* del *token* JWT, de manera que no será necesario un acceso a base de datos en cada verificación.

4.2. Diseño de la arquitectura del sistema

La prueba de concepto se compone de una parte cliente y una parte servidor, compuesta por un módulo de autenticación, un módulo de autorización y un módulo para obtener los recursos, en este caso libros. A continuación, se detalla el diseño de cada una de las partes y se presenta un diagrama de arquitectura mostrando la comunicación entre ambas partes.

Diagrama de arquitectura

A continuación, se muestra el diagrama de arquitectura de la prueba de concepto donde se detalla como se comunican cada uno de los módulos a alto nivel (Ilustración 13).

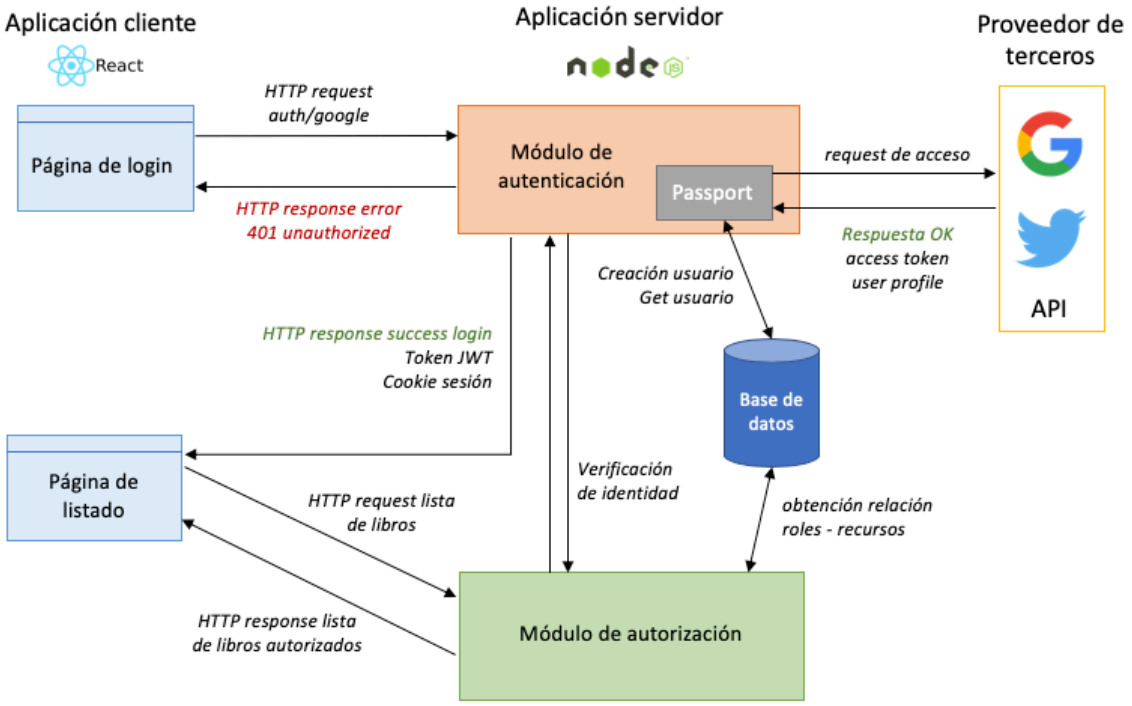

Ilustración 13 Diagrama de arquitectura

Parte cliente

La parte cliente es una *Single Page Application* (SPA) desarrollada en *Reactjs* [14]. La aplicación se comunica con la parte servidor mediante peticiones HTTP a los servicios de autenticación y de acceso a los recursos. En la prueba de concepto, los recursos serán libros almacenados en una base de datos y, según los privilegios, se visualizarán un conjunto de ellos y cierta información de estos.

La aplicación dispondrá de las siguientes páginas:

- Página de autenticación: Tendrá dos cajas de texto para introducir las credenciales de usuario. Por otro lado, dispondrá de 3 botones que permitirán acceder utilizando la identidad proporcionada por terceras partes, concretamente por Google y Twitter.

- Página de listado de libros: El usuario visualizará un listado de libros a los que tiene acceso según sus privilegios.

- Página de detalle de libro: Se mostrará información del libro. Según los permisos del usuario, se visualizará más o menos información del libro.

La autenticación del usuario se realizará mediante el módulo de autenticación que, una vez verificada la identidad del usuario, devolverá un *token* JWT que será almacenado en la sesión del navegador. Mediante un interceptor, se añadirá una cabecera con el *token* de autenticación en el resto de peticiones al servidor. Este *token* contendrá la identidad del usuario y sus privilegios.
A continuación, se muestran 2 diagramas de interacción entre la aplicación cliente y el módulo de autenticación.

El primer diagrama (Ilustración 14), corresponde a los pasos para realizar la autenticación mediante credenciales desde el formulario de autenticación.

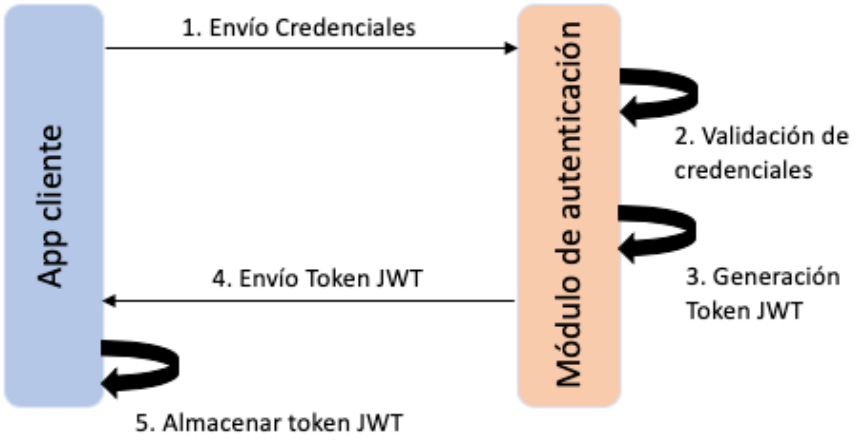

Ilustración 14 Diagrama de autenticación con credenciales

El segundo diagrama (Ilustración 15), corresponde a los pasos para realizar la autenticación utilizando una identidad federada, como por ejemplo Google o Twitter.

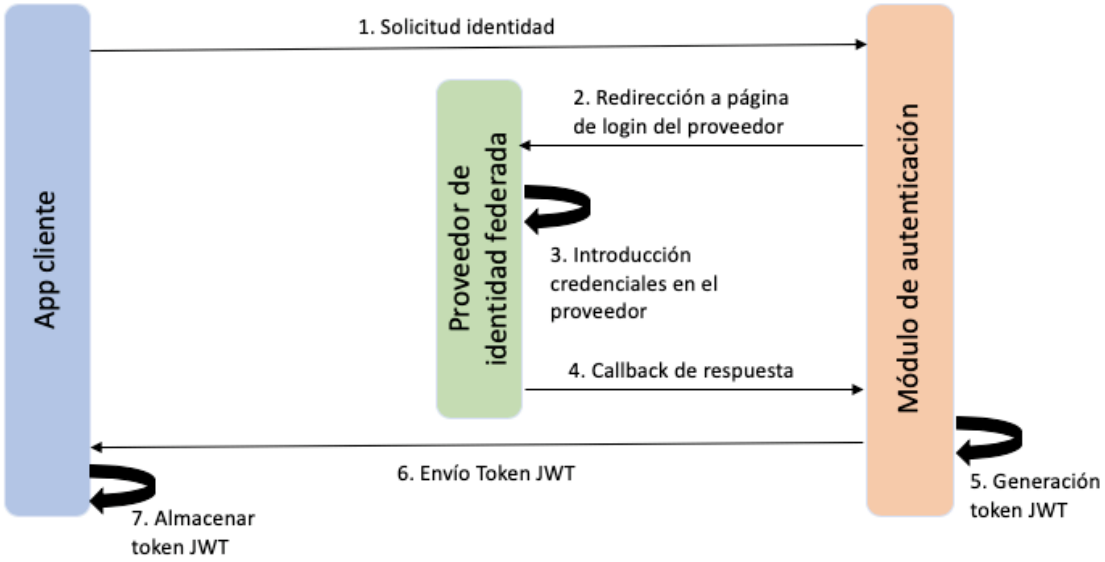

Ilustración 15 Diagrama de autenticación con un proveedor de identidad

Las peticiones a recursos, en este caso libros, se realizarán al servidor que gestiona los libros. Este módulo verificará previamente el token obtenido utilizando el módulo de autenticación. Más adelante se detallará su interacción.

Parte servidor

La parte servidor estará dividida en dos módulos:

- **Módulo de autenticación**

  Este módulo permitirá dos tipos de autenticaciones: mediante credenciales o mediante identidad federada de terceras partes. Este módulo será el encargado de verificar que el usuario es quién dice ser verificando los datos de la petición.

  Una vez validada la identidad del usuario, responderá a la petición con una cookie de sesión y un *token* JWT. El *token* contendrá información básica del usuario y los roles y permisos que tiene asignados.

  Cuando se realicen peticiones a recursos, se delegará la verificación de la identidad del usuario a este módulo. Se utilizará autenticación siguiendo el esquema *Bearer*, por lo que todas las peticiones deberán contener una cabecera de autenticación con el *token* JWT como *Bearer Token*.

En caso de ser un *token* válido, el módulo dejará continuar la petición respondiendo al módulo de autorización que es un *token* válido. En caso contrario, deberá responder con un error HTTP "*401 Unauthorized*".

- **Módulo de recursos (libros)**

  Este módulo será el encargado de gestionar las peticiones a libros que realice la parte cliente. Cuando reciba una petición, realizará los siguientes pasos:

  1. Verificar que existe un *token* JWT en la cabecera de autenticación.
  2. Realizar una petición al módulo de autenticación para verificar si el *token* recibido es válido.
  3. En caso afirmativo, solicitar al módulo de autorización si el usuario tiene privilegios para obtener los recursos solicitados.
  4. En caso afirmativo, acceder a base de datos y devolverá los recursos solicitados, en este caso libros.
  5. Si, por algún motivo, el *token* no es válido o no tiene permisos, se devolverá un error HTTP informativo al cliente.

A continuación (Ilustración 16), se muestra un diagrama con los pasos desde que la aplicación cliente solicita un recurso al módulo de recursos hasta que es devuelto al cliente.

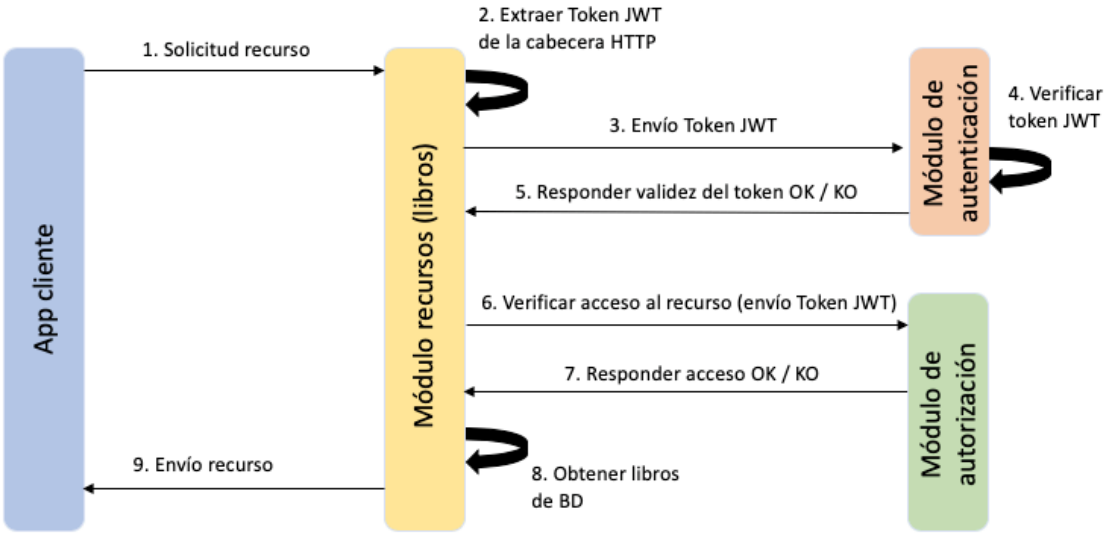

Ilustración 16 Diagrama de solicitud de un recurso

- **Módulo de autorización**

  Este módulo será el encargado de verificar que el usuario tiene los privilegios necesarios para acceder al recurso solicitado. Al recibir una petición, debe extraer los roles y permisos del usuario contenidos en el *token* JWT.

  Una vez disponga de los roles del usuario, debe verificar si dispone de permisos suficientes comparando con la relación rol – recurso almacenados en una base de datos relacional. A continuación, responderá la petición con el resultado de esta verificación.

5. Conclusiones

En el presente trabajo se ha analizado la evolución que han sufrido los sistemas de autenticación y autorización debido a la mayor necesidad de compartir recursos de manera segura en entornos distribuidos. En este estudio, se ha aprendido como inicialmente se idearon sistemas de autenticación pensados solo para una única aplicación y, debido a la necesidad que estos sistemas se comuniquen entre ellos de manera segura, surgieron sistemas que disponen de la autenticación centralizada o gestión de identidad federada.

Se ha logrado uno de los objetivos del trabajo que trataba de analizar como funcionan los sistemas de autenticación única (SSO), la gestión de identidades federadas y los estándares más utilizados en la actualidad.

Otro de los objetivos del trabajo era conseguir diseñar y desarrollar un prototipo que simulara una autenticación y autorización en un entorno distribuido. El análisis en profundidad realizado anteriormente ha permitido diseñar un prototipo de un sistema desde cero que permitiera autenticarse y autorizar recursos utilizando algunos de los estándares anteriormente estudiados.

El prototipo tiene la mayoría de los requerimientos marcados al inicio, pero todavía tiene puntos de mejora que han quedado pendientes para un trabajo futuro. A nivel de seguridad, sería necesario dotarlo de más protección a ataques comunes en el entorno web, pero estos quedan fuera del alcance del proyecto.

La metodología evolutiva / ágil utilizada para realizar el trabajo ha sido la correcta. Ha permitido iterar la solución en cada entrega, desde una versión muy inicial a una con más funcionalidades hasta alcanzar la mayoría de los objetivos propuestos. Respecto a los objetivos iniciales, que eran bastante ambiciosos, se han omitido o simplificado a lo largo del proyecto para obtener un enfoque más realista del trabajo.

Las tecnologías utilizadas para el desarrollo del proyecto son muy nuevas y están en continua evolución. Esto abre otra línea de trabajo futuro, adaptando el sistema a las nuevas versiones de estas tecnologías y protegiéndolo de nuevas vulnerabilidades que van apareciendo. Poco a poco, parece que la gestión de identidades federadas se está imponiendo sobre otros tipos de gestión de identidad, pero será necesario seguir analizando su evolución en el futuro.

Por otro lado, otra línea de trabajo futuro sería la de diseñar el mismo sistema utilizando otras tecnologías diferentes para poder compararlas entre ellas y ver las ventajas e inconvenientes que pueden presentar en un sistema real.

6. Glosario

- **A256GCM**: Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM)
- **AAD**: Additional Authentication Data
- **ACL**: Access Control List
- **AEAD**: Authenticated Encryption with Associated Data
- **API**: Application Programming Interface
- **CEK**: Content Encryption Key
- **DAC**: Discretionary access control
- **DoS**: Denial of service attack (denegación de servicio)
- **E-SSO**: Enterprise SSO
- **HTTP**: Hypertext Transfer Protocol
- **IAM**: Identity and Access Management
- **IdP**: Identity provider
- **JOSE**: Javascript Object Signing and Encryption
- **JWE**: JSON Web Encryption
- **JWS**: JSON Web Signature
- **JWT**: JSON Web Token
- **KDC**: Key Distribution Center
- **LDAP**: Lightweight Directory Access Protocol
- **MAC**: Mandatory access control
- **OIDC**: OpenID Connect
- **PSD2**: Payment Service Directive
- **RBAC:** Role-based access control
- **SAML**: Security Assertion Markup Language
- **SPA**: Single Page Application
- **SSL**: Secure Socket Layer (capa de puertos seguros)
- **SSO**: Single Sign-On
- **ST**: Service Tickets
- **STS**: Security Token Service
- **TGT**: Ticket-granting ticket
- **TLS**: Transport Layer Security (Seguridad de la Capa de Transporte)
- **TPA**: Third Party Application
- **WAM:** Web Access Management
- **XRI:** eXtensible Resource Identifier

7. Bibliografía

[1] *BrowserID* [Fecha de consulta: 26 d'octubre de 2019] <https://github.com/mozilla/id-specs/blob/prod/browserid/index.md>

[2] *Estándar JSON Web Signature (JWS)* [Fecha de consulta: 26 de octubre de 2019] <https://tools.ietf.org/html/rfc7515/>

[3] *Estándar JSON Web Token (JWT)* [Fecha de consulta: 26 de octubre de 2019] <https://jwt.io/introduction/>

[4] *Kerberos: The Network Authentication Protocol* [Fecha de consulta: 26 de octubre de 2019] <https://web.mit.edu/kerberos/>

[5] Kreutz Diego, Luzeiro Feitosa Eduardo, Bessani Alysson, Cunha Hugo (2014, noviembre). *Towards Secure and Dependable Authentication and Authorization Infrastructures.* Singapore: Conference 20th Pacific Rim International Symposium on Dependable Computing – PRDC 2014.

[6] *OAuth 2.0* [Fecha de consulta: 25 d'octubre de 2019] <https://oauth.net/2/>

[7] *OpenID Foundation website* [Fecha de consulta: 25 de octubre de 2019] <https://openid.net/>

[8] *Remote Authentication Dial In User Service (RADIUS)* [Fecha de consulta: 27 d'octubre de 2019] <https://tools.ietf.org/html/rfc2865/>

[9] Rios Joseph, Smith Irene, Venkatesen Priya (2019, septiembre) *UAS Service Supplier Framework for Authentication and Authorization*. California: NASA.

[10] *SAML Specifications* [Fecha de consulta: 26 de octubre de 2019] <http://saml.xml.org/saml-specifications>

[11] *Shibboleth Identity Provider* [Fecha de consulta: 25 de octubre de 2019] <https://www.shibboleth.net/products/identity-provider/>

[12] Umer Khalid, Abdul Ghafoor, Misbah Irum, Muhammad Awais Shibli (2013) *Cloud based Secure and Privacy Enhanced Authentication & Authorization Protocol*. Pakistan: Elseiver B.V.

[13] V Radha, D. Hitha Reddy (2011) *A Survey on Single Sign-On Thecniques*. India: Elsevier Ltd. Selection.

[14] *Documentación de Reactjs* [Fecha de consulta: 27 de octubre de 2019] <https://reactjs.org/>

[15] *Documentación del repositorio Github: create-react-app* [Fecha de consulta: 28 de octubre de 2019] <https://github.com/facebook/create-react-app>
[16] *Documentación Material-UI* [Fecha de consulta: 28 de octubre de 2019] <https://material-ui.com/>

[17] *Documentación de Nodejs* [Fecha de consulta: 28 de octubre de 2019] <https://nodejs.org/en/>

[18] *Documentación de la librería Passport.js* [Fecha de consulta: 28 de octubre de 2019] <http://www.passportjs.org/>

[19] Bokefode Jayant. D., Ubale Swapnaja A., Apte Sulabha S., Modani Dattatray G. (2014, octubre) *Analysis of DAC MAC RBAC Access Control based Models for Security*. India: International Journal of Computer Applications (0975- 8887, Volumen 104 – No 5).

[20] *Criptografía simétrica* [Fecha de consulta: 15 de noviembre de 2019] < https://es.wikipedia.org/wiki/Criptograf%C3%ADa_sim%C3%A9trica>

[21] *Criptografía asimétrica* [Fecha de consulta: 15 de noviembre de 2019] <https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica>

[22] *Transport Layer Security* [Fecha de consulta: 15 de noviembre de 2019] <https://es.wikipedia.org/wiki/Transport_Layer_Security>

[23] *Cookies + Session VS JSON Web Tokens* [Fecha de consulta: 20 de noviembre de 2019] <https://programacionymas.com/blog/jwt-vs-cookies-y-sesiones>

[24] *¿Cómo saber si tienes un JWS o JWE?* [Fecha de consulta: 27 de noviembre de 2019] <https://riptutorial.com/es/jwt/example/18898/-como-saber-si-tienes-un-jws-o-jwe->

[25] *JWT, JWS and JWE for Not So Dummies! (Part I)* [Fecha de consulta: 27 de noviembre de 2019] <https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3#39b4>

[26] *Estándar JSON Web Encryption (JWE)* [Fecha de consulta: 27 de noviembre de 2019] <https://tools.ietf.org/html/rfc7516/>

[27] *Identidad digital* [Fecha de consulta: 5 de diciembre de 2019] <https://www.arimetrics.com/glosario-digital/identidad-digital>

[28] *Identidad digital* [Fecha de consulta: 5 de diciembre de 2019] <https://moodle2017-18.ua.es/moodle/pluginfile.php/39736/mod_resource/content/7/identidad/page_03.htm>

[29] Martínez-Ballesté Antoni, Solanas Agustí, Castellà-Roca Jordi. *Identificación, autenticación y control de acceso*. UOC. Módulo 1 de identidad digital.

[30] *SSO Guide* [Fecha de consulta: 12 de diciembre de 2019] <https://www.getkisi.com/courses/sso-guide>

[31] *Single Sign-On* [Fecha de consulta: 12 de diciembre de 2019] <https://en.wikipedia.org/wiki/Single_sign-on>

[32] L. Jean Camp. (2004, febrero) *Digital Itendity*. IEEE Technology and Society Magazine.

[33] *What is the Web SSO?* [Fecha de consulta: 15 de diciembre de 2019] < https://www.evidian.com/products/web-sso/what-is-the-web-sso/>

[34] *Using a Load Balancer as a Free Reverse Proxy* [Fecha de consulta: 15 de diciembre de 2019] <https://freeloadbalancer.com/using-a-load-balancer-as-a-free-reverse-proxy/>

[35] Steiner Jennifer G., Neuman Clifford, Schiller Jeffrey I. (1988, enero) *Kerberos: An Authentication Service for Open Network Systems*.

[36] *Tickets de Kerberos: Comprensión y explotación* [Fecha de consulta: 15 de diciembre de 2019] <https://www.tarlogic.com/blog/tickets-de-kerberos-explotacion/>

[37] *Federated Identity Pattern* [Fecha de consulta: 20 de diciembre de 2019] <https://docs.microsoft.com/es-es/azure/architecture/patterns/federated-identity>

[38] *Welcome to OpenID Connect* [Fecha de consulta: 20 de diciembre de 2019] <https://openid.net/connect/>

[39] The Authorization Code Flow in Detail [Fecha de consulta: 22 de diciembre de 2019] <https://rograce.github.io/openid-connect-documentation/explore_auth_code_flow>

[40] *Una introducción a OAuth 2* [Fecha de consulta: 27 de diciembre de 2019] <https://www.digitalocean.com/community/tutorials/una-introduccion-a-oauth-2-es>

[41] *Implicit Flow* [Fecha de consulta: 27 de diciembre de 2019] <https://auth0.com/docs/flows/concepts/implicit>

8. Anexos

8.1. Estructura de la aplicación cliente

En este apartado se detalla la arquitectura front-end seleccionada para desarrollar la parte cliente de la prueba de concepto.

El esqueleto inicial de la aplicación se ha generado utilizando la utilidad “create react app” [15], que nos proporciona una estructura básica para empezar a desarrollar la aplicación web en *Reactjs*.

Para diseñar e implementar las pantallas se utilizará la librería de componentes *Material-UI* [16], que nos proporciona una colección de componentes fácil de utilizar y totalmente configurables.

La estructura de carpetas de la aplicación estará organizada de la siguiente manera:

- **actions**: Contiene las *actions* necesarias para interactuar con el estado global de la aplicación (*Redux*) y realizar las peticiones a servidor.

- **assets**: Recursos como imágenes, fuentes, etc.

- **components**: Contiene los componentes de la aplicación.

- **containers**: Contiene los contenedores de componentes, como por ejemplo las páginas anteriormente descritas.

- **reducers**: Contiene los métodos para especificar cómo cambia el estado de la aplicación en respuesta a lo que mandan las *actions* al *store* de la aplicación.

- **package.json**: Define el nombre del paquete y sus dependencias, autor, licencia y otros.

8.2. Estructura de la aplicación servidor

La parte servidor está dividida en varios módulos: uno para la autenticación, otro para la autorización y otro para solicitar recursos, en este caso libros. Para poder reutilizar el módulo de autenticación y autorización en un sistema distribuido, se ha utilizado una arquitectura de microservicios con un microservicio para cada módulo.

Se implementarán en *Nodejs* [17] y *Expressjs*, debido a que está muy orientado a este tipo de arquitectura y que dispone de utilidades para trabajar con *tokens* JWT y librerías de autenticación. Para poder utilizar identidad, se utilizará la librería *Passport.js* [18], que dispone de módulos compatibles con los principales proveedores de identidad.