Universitat Oberta de Catalunya

uoc.edu

TRABAJO FINAL DE MÁSTER

WebAuthn

Autor: Alberto Gabriel Ruiz Pérez

Máster Interuniversitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC)

Sistemas de autenticación y autorización

Nombre Consultor: Pau del Canto Rodrigo
Nombre Profesor/a responsable de la asignatura: Víctor García Font

Junio, 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*WebAuthn*
Nombre del autor:	*Alberto Gabriel Ruiz Pérez*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	*Máster Interuniversitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC)*
Área del Trabajo Final:	*Sistemas de autenticación y autorización*
Idioma del trabajo:	*Español*
Palabras clave	*Estándar, autenticación web, criptografía asimétrica*

Resumen del Trabajo

Con el paso del tiempo podemos darnos cuenta como el número de servicios que utilizados aumentan, además de las respectivas credenciales que debemos recordar para su acceso. Del mismo modo, al tener éstas una gran importancia, son el principal objetivo en los ataques cibernéticos, poniendo en serio peligro la seguridad (integridad, confidencialidad y disponibilidad) del servicio y de los datos de los usuarios.

Así pues, el presente trabajo tiene como finalidad realizar una introducción sobre el nuevo estándar WebAuthn, un protocolo en los navegadores web que pretende proveer a las aplicaciones web de un sistema de autenticación seguro mediante tokens físicos, donde las contraseñas ya no se tengan que emplear y donde las tradicionales vulnerabilidades/ataques no tengan cabida.

Este documento reflejará el trabajo realizado, el cual se dividirá en una serie de fases.

En primer lugar, se procederá a desempeñar la parte teórica del estándar, en el que se comentarán los aspectos técnicos y su forma de funcionamiento, terminando así con los posibles casos de uso.

Finalmente, para un mejor entendimiento del protocolo, se realizará un ejemplo práctico donde pongamos en funcionamiento WebAuthn como método de autenticación en una supuesta aplicación web. Esto nos ayudará a visualizar a mejor su alcance, para que desde nuestro punto de vista podamos obtener las ventajas y desventajas que le rodea.

Abstract

Over time we can realize how the number of services we have increased, in addition to the respective credentials that we must remember to access. In the same way, since they are of great importance, they are the main target in cyber attacks, seriously endangering the security (integrity, confidentiality and availability) of the service and user data.

Thus, this paper aims to make an introduction to the new WebAuthn standard, a protocol in web browsers that aims to provide web applications with a secure authentication system using physical tokens, where passwords no longer have to be used. and where traditional vulnerabilities / attacks haven't place.

This document will reflect the work done, which will be divided into a series of phases.

In the first place, the theoretical part of the standard will be carried out, in which the technical aspects and their way of functioning will be discussed, ending with the possible use cases.

Finally, for a better understanding of the protocol, a practical example will be made where we put WebAuthn into operation as an authentication method in a supposed web application. This will help us to better visualize its scope, so that from our point of view we can obtain the advantages and disadvantages that surrounds it.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Conforme avanza la tecnología y en especial la que rodea a las tecnologías web, vemos como el número de servicios a través de internet crece (banca, ocio, redes sociales, etc) y con ello el número de credenciales a recordar. Se hace prácticamente imposible mantener segura nuestra identidad, ya que lógicamente también aumenta la probabilidad de amenazas y debilidades sobre sus plataformas que pongan en peligro nuestra información.

Y es que para afirmar lo dicho, vemos como actualmente aumenta la cantidad de noticias que nos informan sobre casos de robos y divulgación de credenciales a las plataformas más famosas y que más usuarios alojan.

Un ejemplo de ello podría ser la catalogada como “Collection #1” [1], nombre que le han asignado a la que en fecha de enero de 2019 dicen que es la mayor filtración de la historia, en la que se han visto comprometidas 773 millones de correos electrónicos en cuentas de multitud de servicios, lo que supone un número mayor de combinaciones de credenciales (usuario-contraseña).

Además, una empresa referente en la seguridad informática como es Karpersky, afirma que en los últimos años los ataques a la banca electrónica van en aumento [2], representando un 15,9% en comparación del año 2018 respecto a 2017.

Lo lógico es que en las plataformas más usadas/importantes se implanten las mejores medidas de seguridad, pero al ser el principal foco de atención mantener una seguridad plena se convierte en una utopía.
El robo de credenciales podría suponer por tanto multitud de acciones malintencionadas hacia el usuario, como por ejemplo la suplantación de identidad, robo de recursos económicos, información valiosa y un largo etcétera que todo podemos imaginar.

Así pues, tal y como se dijo anteriormente, este Trabajo Final de Máster se enmarca en lo relativo a las tecnologías web y especialmente en analizar y detallar el estado actual y futuro de un nuevo estándar, WebAuthn, que nace con la principal intención de poner fin al uso de las contraseñas [3], y con ello a las tantísimas amenazas que figuran sobre ellas.

A pesar de que se detallará de mejor manera en capítulos posteriores, de forma introductoria y muy general podríamos decir que WebAuthn ayudará a las páginas web a incluir un nuevo método de registro/autenticación, donde la novedad radica en la utilización de tokens físicos que funcionarán con la famosa criptografía asimétrica (clave pública).

1.2 Objetivos del Trabajo

Una vez comentado el tema principal de este Trabajo Final de Máster, nos hemos propuesto una serie de objetivos a cumplir en el periodo de tiempo establecido. A continuación se listarán los fundamentales, los cuales se irán incluyendo posteriormente en las distintas fases definidas en la planificación:

- El primer y principal objetivo a grandes rasgos será familiarizarnos, entender y testear el nuevo estándar WebAuthn, de esta manera posteriormente podremos obtener nuestras propias conclusiones de una forma más precisa y fiable.
- Para entender el propósito de la creación de este estándar, deberemos de detallar las causas (problemas con métodos anteriores, amenazas, etc) que han llevado a su creación, así como también los beneficios e inconvenientes que obtenemos con su implantación y uso.
- Realizar un repaso de los métodos de autenticación existentes, centrándonos (debilidades) en las características de seguridad que posee.
- Detallar las especificaciones del estándar WebAuthn, los protocolos de comunicación que utiliza y los posibles casos de uso.
- Debido a que este estándar requiere el uso de tokens físicos de seguridad que mediante protocolos específicos se comuniquen con la API (WebAuthn), se realizará un repaso por el mercado actual para identificar si se está apoyando esta iniciativa y cuales son los requerimientos en cuanto a dispositivos nos referimos (marcas, precios, etc).
- De igual manera que el anterior objetivo, se estudiará además el nivel de aceptación de dicha API por los navegadores más utilizados, ya que juegan un papel fundamental para que el proceso de estandarización se lleve a cabo y sea conocido/implementado por los desarrolladores.
- Por último, otro de los objetivos fundamentales es poner en práctica la teoría de los anteriores objetivos, es decir, incorporaremos la utilización de WebAuthn como método de registro/autenticación a una supuesta página web (tanto backend como frontend), de tal forma que podamos ver y comprobar de primera mano tanto el nivel de dificultad que supone su integración como el nivel de seguridad que aporta a los usuarios finales.

Así pues, los objetivos comentados guiarán continuamente el proyecto y los procesos a ejecutar.

1.3 Enfoque y método seguido

El presente trabajo, podríamos decir que tiene un enfoque mixto, ya que por un lado pretendemos establecer una toma de contacto teórica con el nuevo estándar WebAuthn, para que posteriormente nos introduzcamos en un entorno de desarrollo poniendo en práctica lo aprendido. Por este motivo, tal y como se puede apreciar hemos dividido el índice de este documento en dos partes bien diferenciadas (teórico y práctico), ya que a pesar de estar claramente

relacionadas, las acciones que se llevan a cabo en cada una de ellas son bien diferentes.

Por otro lado, para este proyecto no hemos aplicado ninguna metodología específica, sino que tanto la finalidad y planificación de tareas han sido acordadas con el tutor del TFM, que por medio de una comunicación y revisión continuas se ha guiado la consecución de los objetivos definidos.

Así pues, tal y como anteriormente los listábamos (objetivos principales), éstos se irán concluyendo a medida que el proyecto avanza. Así pues, se ha dividido en una serie de fases, cada una de ellas con un alcance bien definido:

- **1ª Fase: Plan de trabajo**: En esta primera etapa se realizará una contextualización y justificación de este proyecto. Además, definiremos los objetivos y la metodología para cumplir el plan previsto, que tal y como vemos en este momento, será dividido en tareas a ejecutar en el tiempo.

- **2ª Fase: Desarrollo del marco teórico sobre el estándar WebAuthn**: Durante esta segunda etapa nos introduciremos en el interior de WebAuthn, en su especificación para comprender de manera correcta su modo de funcionamiento y sus principales características.

- **3ª Fase: Implantación de WebAuthn en una supuesta página web**: Una vez hayamos comprendido fielmente el estándar, culminaremos el proyecto con la puesta en marcha en una supuesta página web de un nuevo método de autenticación, haciendo uso de la API WebAuthn. Con ello comprobaremos de mejor manera los procesos que se llevan a cabo, sacando nuestras propias conclusiones desde los diferentes puntos de vista, teórico, desarrollador y usuario.

- **4ª Fase: Desarrollo del documento de la memoria del TFM**: A pesar de que se ha posicionado en este orden, veremos en diagramas posteriores como esta fase se ejecuta durante todo el tiempo. En este documento se incluirá toda la información recogida en las diversas fases comentadas (conclusiones detectadas, teoría, etc).

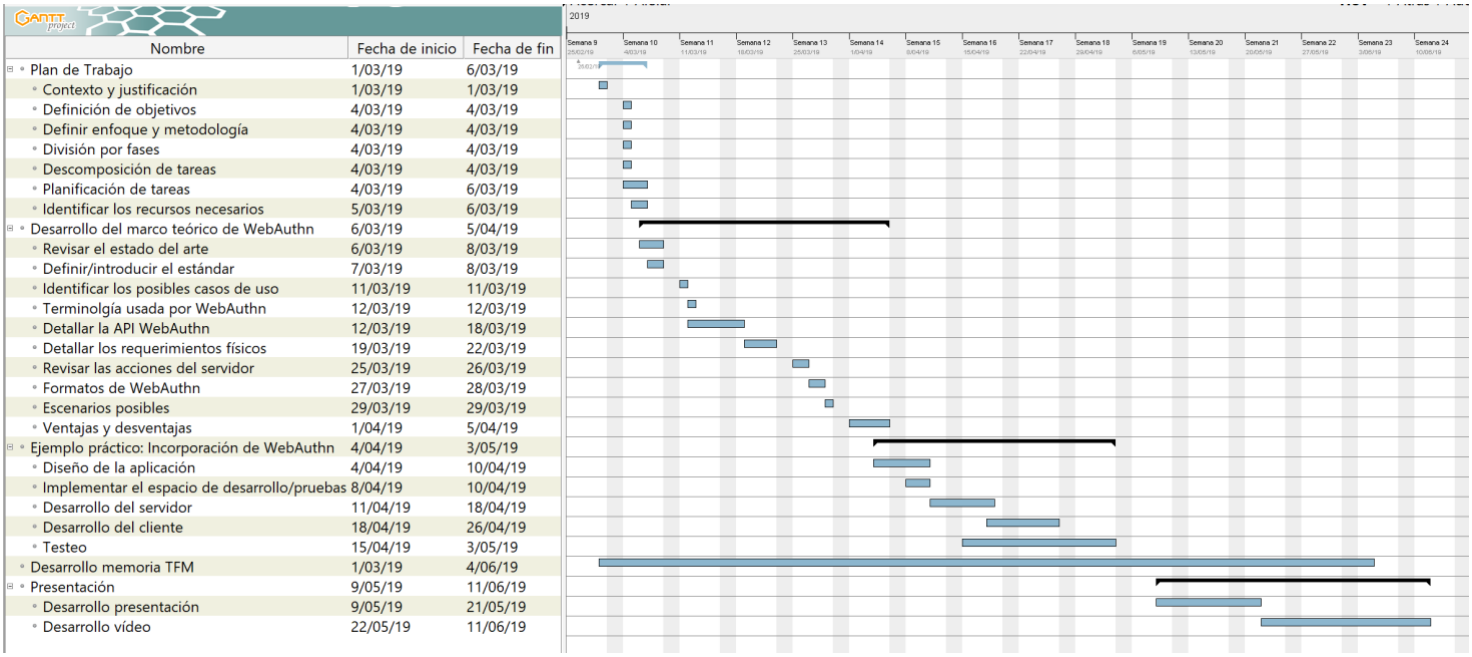
- **5ª Fase: Presentación del proyecto y los resultados obtenidos**: Por último, realizaremos los documentos y vídeos de presentación correspondientes, demostrando así lo realizado en este proyecto.

1.4 Planificación del Trabajo

En este apartado englobaremos todo lo relativo a la planificación del trabajo, descompondremos las fases anteriormente descritas en acciones más específicas (tareas), posicionándolas a su vez en el espacio de tiempo establecido (diagrama de Gantt). Por otro lado, comentaremos los recursos que serán necesarios para el proyecto.

1.4.1 Planificación de tareas y división en el tiempo

Para una mejor especificación y planificación de tareas, en este apartado desarrollaremos el famoso diagrama de Gantt, en el que descomponemos las fases establecidas en acciones de menor alcance, indicándole además el periodo de tiempo que se le dedicará.



Nombre	Fecha de inicio	Fecha de fin
Plan de Trabajo	1/03/19	6/03/19
Contexto y justificación	1/03/19	1/03/19
Definición de objetivos	4/03/19	4/03/19
Definir enfoque y metodología	4/03/19	4/03/19
División por fases	4/03/19	4/03/19
Descomposición de tareas	4/03/19	4/03/19
Planificación de tareas	4/03/19	6/03/19
Identificar los recursos necesarios	5/03/19	6/03/19
Desarrollo del marco teórico de WebAuthn	6/03/19	5/04/19
Revisar el estado del arte	6/03/19	8/03/19
Definir/introducir el estándar	7/03/19	8/03/19
Identificar los posibles casos de uso	11/03/19	11/03/19
Terminolgía usada por WebAuthn	12/03/19	12/03/19
Detallar la API WebAuthn	12/03/19	18/03/19
Detallar los requerimientos físicos	19/03/19	22/03/19
Revisar las acciones del servidor	25/03/19	26/03/19
Formatos de WebAuthn	27/03/19	28/03/19
Escenarios posibles	29/03/19	29/03/19
Ventajas y desventajas	1/04/19	5/04/19
Ejemplo práctico: Incorporación de WebAuthn	4/04/19	3/05/19
Diseño de la aplicación	4/04/19	10/04/19
Implementar el espacio de desarrollo/pruebas	8/04/19	10/04/19
Desarrollo del servidor	11/04/19	18/04/19
Desarrollo del cliente	18/04/19	26/04/19
Testeo	15/04/19	3/05/19
Desarrollo memoria TFM	1/03/19	4/06/19
Presentación	9/05/19	11/06/19
Desarrollo presentación	9/05/19	21/05/19
Desarrollo vídeo	22/05/19	11/06/19

Ilustración 1 - Diagrama de Gantt

Así pues, tal y como se puede visualizar en la ilustración anterior, se han descompuesto las fases establecidas en tareas de menor tamaño y mayor especificación. Además, si nos fijamos en el periodo de tiempo de cada una de ellas, las fases coinciden con los plazos de entrega de la asignatura.

Un aspecto importante que debemos tener en cuenta es la dependencia de tareas, es decir, la importancia del orden en que son ejecutadas o solapadas. Como vemos, la gran mayoría de ellas son realizadas de manera secuencial, y es que como es lógico, la información obtenida en cada una se utilizará para enriquecer a las posteriores. Un ejemplo de ello podría ser las tareas del marco teórico, ya que en primer lugar deberemos de informarnos sobre las características del estándar para luego definir nuestras propias conclusiones (escenarios posibles, ventajas y desventajas, etc). Sin embargo, también habrá tareas que se ejecuten de manera paralela, como es el caso de la generación del documento de la memoria, que será completada conforme se avanzan en los objetivos y tareas.

1.4.2 Recursos necesarios

En este apartado comentaremos los recursos necesarios para realizar las tareas anteriormente comentadas y completar así los objetivos definidos:

- **Software de edición de documentos**: Para la generación de los diferentes documentos que han de entregarse, ha sido necesario el empleo de programas de edición de texto y presentaciones, en nuestro caso hemos escogido el grupo de aplicaciones de Microsoft Office (Word, Power Point, etc) en su versión 2019.

- **Software de captura de vídeo:** Tal y como se indicó en la planificación de tareas, una vez terminado se realizará una presentación sobre el trabajo realizado. Para ello deberemos capturar la pantalla y sonido del equipo, ya que así podremos visualizar la presentación y demostrar la aplicación desarrollada. Así pues, hemos escogido el software de código abierto OBS studio.

- **Software de planificación de tareas:** Como se ha podido ver en la planificación del trabajo, se ha descompuesto cada fase en tareas más detalladas que se ejecutan en un tiempo determinado. Para poder visualizar de mejor manera el orden y los casos de solapamiento, se ha realizado un diagrama de Gantt (Ilustración 1), por lo que ha sido necesario un software para tal fin, en este caso la aplicación de código abierto GanttProject en su versión 2.8.10 [4].

- **Entorno de desarrollo:** Este proyecto tiene un enfoque mixto, ya que además de hacer un repaso teórico sobre el estándar WebAuthn realizaremos un desarrollo de prueba, que consistirá en la implantación de este estándar como método de autenticación en una supuesta página web. Así pues, lógicamente será necesario establecer un entorno adecuado, con ello nos referimos a la instalación de las tecnologías necesarias (dependiendo de las que se escojan más adelante: Python, java, nodejs, php, librerías, etc), sistemas de virtualización, navegadores para realizar pruebas (Chrome, Firefox, Safari, Explorer, etc), IDEs o editores de texto (Netbeans, Eclipse, Atom, Visual Basic, CodeBlocks), etc.

- **Tokens físicos autenticadores:** A pesar de que llegados a este punto todavía no se ha comentado mucho sobre WebAuthn, ya se dijo en la introducción que éste estándar hace uso de tokens físicos y que éstos a su vez emplean criptografía asimétrica. Por lo tanto, para realizar la parte más práctica de este trabajo y testear la aplicación desarrollada, será imprescindible disponer de estos dispositivos. Cabe destacar que deberán soportar una serie de protocolos específicos, como son U2F o el actualizado FIDO2. Así pues, en nuestro caso hemos decidido obtener dos tokens de marcas y protocolos diferentes:

    - **Yubyco Security Key**: Clave de seguridad que combina la autenticación basada en hardware, la criptografía de clave pública y los protocolos U2F y FIDO2 [5].

Ilustración 2 - Yubyco Security Key

- **HYPERSECU HyperFIDO Mini:** Clave certificada de seguridad FIDO U2F, que proporciona compatibilidad total con los sistemas operativos más usados y sin limite de almacenamiento de pares de claves o cuentas de usuarios.

Ilustración 3 - HYPERSECU HyperFIDO Mini

Por último, el motivo por el cual se ha decidido hacer uso de dispositivos con características y funcionalidades diferentes es que nos servirá para mostrar las diferencias de adaptación de WebAuthn sobre ellos (protocolos U2F y FIDO2). De igual manera, esto nos ayudará a revisar el mercado actual para comprobar el nivel de apoyo hacia este tipo de sistemas.

1.5 Breve sumario de productos obtenidos

Del mismo modo que en apartados anteriores dividíamos la realización del proyecto en diferentes fases, en cada una de ellas obtendremos una serie de entregables que coincidirán con las respectivas entregas de la asignatura:

- **PEC 1:** En esta primera entrega, se realizará el documento correspondiente al plan de trabajo. En él se definirá una breve introducción y contextualización sobre el proyecto a realizar, además de la panificación que se llevará.

- **PEC 2:** La segunda entrega corresponde con la consecución de las tareas existentes dentro del mismo espacio de tiempo. Tal y como se dijo, la generación del documento de la memoria se realizará desde el primer momento, incluyendo cada uno de los avances que se vayan

completando. Así pues, en este caso estaríamos hablando del marco teórico del estándar WebAuthn, la primera parte de este proyecto (capítulos 1,2 y 3), donde se definen sus características y funcionalidades.

- **PEC 3:** En este caso, una vez entendida y documentada la parte teórica del proyecto, esta entrega será todo aquello relacionado con la parte práctica, la implantación de WebAuthn en una supuesta página web. Del mismo modo que en la entrega anterior, iremos completando el documento de la memoria con la información obtenida durante esta fase.

- **PEC 4:** En estas últimas fases del proyecto, se culminará el documento de la memoria del TFM, donde además de la información anteriormente descrita se incluyen las conclusiones obtenidas, el trabajo futuro, bibliografía consultada, etc.

- **PEC 5:** En la última entrega, correspondiente a la presentación del trabajo realizado, los entregables serán tanto el documento de presentación como el vídeo demostrativo.

1.6 Breve descripción de los otros capítulos de la memoria

Una vez introducido, planificado y puesto en contexto el proyecto, en los siguientes capítulos nos introduciremos en todo lo referente a WebAuthn.

En primer lugar, a continuación realizaremos una revisión (estado del arte) de los métodos de autenticación/registro existentes y más usados, mostrando sus ventajas y desventajas y recalcando los motivos que han llevado a la creación de este nuevo estándar.

Una vez entendamos el por qué de su creación, revisaremos su documentación para obtener las características de su funcionamiento, mostrando los beneficios que obtendremos con su utilización y las posibles debilidades.

Cuando se haya completado la fase teórica del proyecto, se pondrá en práctica lo aprendido sobre el estándar WebAuthn, así pues se implantará como método de autenticación/registro en una página web. Comentaremos tanto las funcionalidades del servidor como del cliente. Esto junto con los tokens físicos nos permitirá testearlo, sabiendo así el nivel de dificultad de implantación y los límites que tiene.

Por último, incluiremos en este documento un glosario de términos relevantes en el trabajo y la bibliografía consultada que nos ha servido de ayuda para obtener información.

2. Estado del arte

Para entender los motivos que han llevado a la creación del estándar WebAuthn, es conveniente que repasemos los antecedentes y el estado del arte que lo envuelve, especialmente en este apartado valoraremos (siempre desde el punto de vista de la seguridad) la evolución de los diferentes métodos de autenticación y registro que son utilizados en la mayoría de plataformas web.

Tal y como comentamos en la introducción de la memoria, vemos como actualmente aumentan el número de casos de robo y filtración de credenciales, las campañas de phishing y todos los métodos para robar información están a la orden del día.
Obviamente, las credenciales tienen un gran valor, ya que con ellas se puede acceder a las respectivas plataformas y realizar todas las acciones que estas posibilita, imaginémonos así la magnitud de acciones que se podrían ejecutar y que influirían negativamente al usuario legítimo.

Siguiendo este contexto, podemos darnos cuenta que con el paso del tiempo los desarrolladores de estas plataformas han intentado en todo momento mejorar la seguridad de sus servicios (en nuestro caso métodos de autenticación) y proteger a los usuarios de las amenazas anteriormente nombradas, así pues a continuación podríamos realizar un breve barrido de algunos de los métodos de autenticación más utilizados en plataformas web:

- **Credencial usuario/contraseña:** Sin duda alguna es el método más conocido y básico que existe, pero a su vez son muchísimas las amenazas que hay sobre ellas. Podríamos decir que para que éstas tengan seguridad, no basta con usar contraseñas 'difíciles de adivinar' ni utilizar un canal de comunicación seguro (SSL), ya que por ejemplo ataques de phishing o MITM podrían poner en riesgo su estado de seguridad.

- **Autenticación al cliente por certificados:** Al igual que se establece una comunicación segura con el servidor (TLS, SSL), se puede realizar un método en el que el servidor le exija al cliente presentar su certificado, cifrando así la comunicación entre ambos puntos y verificando que es el usuario legítimo. Una desventaja de este método podría ser la dificultad y el rendimiento que posee, ya que habría que configurar cada navegador cliente con su respectivo certificado.

- **Métodos de segundo factor (A2F) o multifactor (MFA):** Otro de los métodos que actualmente está siendo más empleado es hacer uso de diferentes factores de autenticación, de tal manera que aumentemos la seguridad y disminuyamos la posibilidad (aumentar la dificultad) de completar un ataque. Entre esos factores podemos nombrar algunos, como por ejemplo algo que se sabe (contraseña, respuesta a una pregunta), algo que se tiene (token de seguridad, código único temporal (TOTP), códigos enviados por sms o email, etc) o algo que es

(características únicas de la persona, como por ejemplo el scaner de retina, huella dactilar, etc).

Como hemos podido apreciar, lógicamente ha habido intención de crear sistemas de autenticación seguros, pero incuso hasta el último punto (A2F y MFA) siguen existiendo factores que pueden llegar a ser amenazados por diversos ataques (intercepción de sms, robo de clave de generación de códigos temporales TOTP, phishing, etc).

De esta manera nace la necesidad de dar un cambio de rumbo en cuanto a la autenticación nos referimos, y es aquí cuando entra en juego el nuevo estándar WebAuthn.
A pesar de que se comentará en profundidad en capítulos posteriores, podemos decir que el objetivo principal es el de poner fin al uso de contraseñas, es decir, con su implantación los usuarios ya no tendrán que enviar información confidencial (contraseña) al servidor, desapareciendo así los vectores de ataque comentados, y todo ello gracias a la inclusión de la criptografía asimétrica en tokens físicos y la estandarización (WebAuthn) en los navegadores.

3. Estándar WebAuthn

World Wide Web Consortium (W3C) y Alianza FIDO publicaron el 4 de marzo de 2019 que el proyecto de Web Authentication pasaría a ser un estándar web oficial, cuyo principal objetivo es hacer una web más segura y utilizable.
Éste estándar de navegador y plataforma, prevé una interfaz para implantar una autenticación más simple y fuerte (haciendo uso de criptografía de clave pública en tokens físicos) que las comentadas en capítulos anteriores.

Así pues, durante este capítulo y tal y como predijimos anteriormente, trataremos el marco teórico referente a la especificación de WebAuthn, dividiéndolo en una serie de apartados para tratar sobre temas diferentes pero relacionados entre ellos, que a medida que avancemos iremos entendiendo las funcionalidades y responsabilidades de cada componente que lo conforma.

3.1 Introducción

Siguiendo la temática anterior, nos referimos a WebAuthn como una estandarización de una API, la cual permite la creación y el uso de credenciales basadas en criptografía asimétrica (pública) para plataformas/aplicaciones web, cuyo propósito es el de proveer un sistema de autenticación/registro seguro, con la posibilidad de no usar contraseñas y con ello eliminar aquellas amenazas que le perseguían.

Su modo de funcionamiento desde una perspectiva general es bastante simple. En principio, por medio de dicha interfaz (api de webauthn en javascript) el agente de usuario (navegador del cliente) podrá enviar acciones hacia el autenticador (el token físico de seguridad, conectado por ejemplo a través de

usb), que será el encargado de crear y almacenar un par de claves (pública y privada, además de otra información). Posteriormente, solamente el origen (dirección de la plataforma, alcance) podrá acceder y hacer uso de dichas claves alojadas en el token.

Básicamente, las que llamamos 'partes confiables' (servidores, página web) podrán emplear la API de WebAuthn para dos tipos de finalidades distintas. Por un lado está el caso de registro, donde la generación de un nuevo par de claves asociado a dicho origen se lleva a cabo, y por otro lado es el caso de la autenticación, donde la plataforma verificará la presencia y consentimiento del usuario, que consistirá en enviar una aserción (desafío) al autenticador, el cual deberá responder con la firma de dicho desafío con la clave privada correspondiente, comprobando así en el lado del servidor si la firma es correcta en base al usuario.

Así pues, los autenticadores/tokens compatibles llevan a cabo una serie de métodos para proteger las credenciales almacenadas e interactuar con los agentes de usuario por medio de la API de WebAuthn. De este modo, existen autenticadores implementados en el mismo equipo (de plataforma) y otros desde un dispositivo (itinerancia), el cual se podrá comunicar por medio de diferentes modos de transporte (USB, BLE o NFC).

Tal y como dijimos, en este método de autenticación no se envía ninguna información secreta (clave privada, contraseña) por la red, lo cual nos proporciona un nivel de seguridad que con otros métodos no conseguíamos.

3.2 Terminología

Como se puede detectar en el índice de la memoria, existen dos capítulos que pueden llegar a parecer idénticos, que son el de terminología (3.3) y el del glosario (6). Lógicamente en ambos se alojarán las definiciones de términos importantes, pero para una mejor diferenciación se ha decidido que en este apartado (terminología) estén solamente aquellos que consideremos relevantes y que pertenezcan al contexto de WebAuthn y a su especificación oficial [8]. De otro modo, en el capítulo del glosario estarán aquellos términos importantes en el contexto general del documento y todo aquello que pueda tener relación con el estándar.

- **Atestación (Attestation):** En el ámbito de WebAuthn, nos referimos a una atestación (testimonio, confirmación) como el proceso/objeto que verifica la procedencia de un autenticador y los datos que emite (AAGUID, claves del fabricante, par de claves del usuario, ID, etc), de esta manera el servidor podrá medir el grado de confianza que deberá depositar en dicho autenticador (se realiza durante la etapa de registro).

- **Certificado de atestación (Attestation Certificate):** Certificado X.509 por el cual el autenticador trata de certificar su fabricación y capacidades. En el momento de generar un nuevo par de credenciales, el autenticador lo firmará con su clave privada de certificación, pudiendo posteriormente verificarlo en el lado del servidor.

- **Aserción (Assertion):** Es el objeto por el cual el autenticador trata de demostrar a la plataforma que el usuario posee el par de claves correspondiente (durante la etapa de autenticación), generalmente firmando un desafío recibido desde el servidor.

- **Ceremonia de autenticación:** Dentro del método de autenticación, se denomina ceremonia de autenticación al proceso y comunicación existente entre el cliente (agente de usuario), usuario y autenticador. En este momento se trata de demostrar que el usuario posee el par de claves correspondiente. Además podría ser posible que el usuario deba pasar una prueba de presencia en el autenticador, como por ejemplo la pulsación de un botón, verificación de huella dactilar, etc.

- **Afirmación de autenticación:** Nos referimos al objeto firmado con las claves asimétricas recibido por el autenticador, habitualmente después del envío de un desafío que compruebe su identidad.

- **Autenticador:** Dispositivo o entidad criptográfica que el usuario hace comunicar con el agente de usuario, y en el que se generan credenciales de clave pública para que posteriormente puedan ser empleadas para firmar desafíos y autenticar al usuario en cuestión frente a un origen/plataforma.

- **Gesto de autorización:** Durante el proceso de comunicación entre el cliente y el autenticador, puede ser posible que se necesite verificar la presencia del usuario, para ello se le exigirá que realice algún gesto (pulsación de botón por ejemplo) como forma de consentimiento.

- **Autenticador biométrico:** Cualquier dispositivo de autenticador que incorpore métodos de verificación biométrica (por ejemplo huella dactilar).

- **Plataforma cliente:** La combinación de un dispositivo junto con la implantación del estándar (API) formará la llamada plataforma cliente.

- **Lado del cliente:** Dentro de los procesos de comunicación, el lado del cliente estará constituido por la plataforma del cliente, usuario y autenticador utilizado.

- **Credencial residente:** Consiste en un par de claves que es almacenada en un autenticador con la propiedad de clave residente. La característica principal es que durante el método de autenticación, el usuario podrá escoger (por ejemplo mediante una lista en pantalla) la credencial con la que autenticarse sobre un ID del servidor. Este procedimiento podrá ser útil para la autenticación de factor único sin contraseña ni nombre de usuario.

- **Agente de usuario:** Denominamos agente de usuario a la entidad que implementa la API de WebAuthn (por ejemplo un navegador web) y que maneja la comunicación entre el servidor y autenticador.

- **ID de credencial:** Cadena de bytes que identifica a un par de claves de manera única (gracias a la utilización de al menos 100 bits de entropía). Es generado cuando se da lugar la creación de unas nuevas claves (registro), y que comúnmente se almacenará en el servidor para solicitar un assertion posterior (a través de su id).

- **Clave pública del usuario:** Nos referimos al par de claves que el autenticador genera. La parte pública de dichas claves será trasladada al servidor durante el proceso de registro, mientras que la parte privada será usada en los momentos de autenticación.

- **Fuente de credenciales de clave pública:** Es el término denominado a la información de la credencial almacenada en el autenticador y que es utilizada para generar aserciones de autenticación.

- **Límite de tasa:** Método utilizado por los autenticadores para controlar los ataques de fuerza bruta. Estos limitan el número de intentos fallidos de forma consecutiva, ejecutando un retraso cuando se considere necesario (se detecte la posibilidad de existencia de un ataque).

- **Ceremonia de registro:** Conjunto de procesos y comunicaciones que se dan lugar en el lado del cliente para generar una credencial de clave pública asociada al origen y usuario.

- **Identificador de la parte que confía (ID RP):** Cadena de texto (login.example.com) que referencia al servidor (parte que confía) y que es asociada a cada par de claves. De esta manera solo el origen podrá hacer uso de aquellas credenciales que posean el mismo ID RP. En este caso el puerto del origen no deberá ser específico, pero sí es necesario que la comunicación sea por un canal seguro (HTTPS).

- **Identificador de usuario (User Handle):** Es una cadena (adosada a la información de una credencial) que es especificada por el servidor y que su valor referencia a la cuenta del usuario (id).

- **Cliente WebAuthn:** Nos referimos con este término a la entidad intermediaria (API de WebAuthn) alojada en el agente de usuario.

- **Dispositivo cliente WebAuthn:** Hardware en el que se ejecuta el cliente de WebAuthn (por ejemplo un móvil, ordenador, etc).

- **Tercera parte que confía (WebAuthn Relying Party):** Nos referimos con este término a la plataforma (aplicación web) que hace uso de la API de WebAuthn para autenticar a sus usuarios.

- **FIDO:** FIDO Alliance, es un consorcio que desarrolla estándares de autenticación sin contraseña, seguros y en contra de la suplantación de identidad (phishing). De esta manera podemos referirnos a una pareja de protocolos, U2F y FIDO2.

- **FIDO2:** Un nuevo protocolo de autenticación sin contraseña, que está formado tanto por WebAuthn (API en el navegador) y CTAP (API en el autenticador).

- **CTAP:** Conjunto de protocolos para comunicarse desde el cliente con el autenticador por diferentes medios de transporte.

- **CTAP2:** Corresponde con la segunda versión del protocolo CTAP. La característica principal es que para la codificación usa CBOR, además tiene compatibilidad hacia atrás con CTAP1 (U2F), extensiones y nuevos formatos de certificación.

Una vez comentados los términos considerados como más relevantes en el estándar WebAuthn, en los próximos apartados podrán ser mencionados para profundizar más en las características y funcionalidades que posee.

3.3 Web Authentication API

Durante esta sección veremos los diferentes procesos y requerimientos que serán necesarios para crear y usar credenciales de clave pública por medio de la API de WebAuthn. Tal y como hemos comentado a lo largo del documento, la funcionalidad de WebAuthn se resume a dos tipos de procedimientos, por un lado el de registro y por otro el de autenticación. Así pues, a continuación dividiremos el apartado en dos subsecciones, donde detallaremos el flujo de comunicación existente en ambos métodos desde el punto de vista de la API de Javascript WebAuthn (cliente).

En primer lugar, considero que es importante indicar que WebAuthn en los navegadores es una extensión de la API de administración de credenciales (Credentials Management API), por lo que la estructura y formatos de los objetos están relacionados.
Desde la propia interfaz de WebAuthn es imposible acceder directamente a las credenciales del autenticador, por ello se nos proporciona una serie de métodos para ‘enviar señales’ al dispositivo y que sea en este donde se realicen todos los métodos. Esta forma de abstracción nos provee además una capa de protección, puesto que solamente la plataforma que haya creado previamente las credenciales podrá hacer uso de ellas, limitando su alcance al dominio utilizado.

3.3.1 Registro

Así pues, cuando un usuario quiera hacer uso de un autenticador como método de autenticación en su cuenta de una plataforma, deberá en primer lugar realizar un registro de credenciales (generar un nuevo par de claves).

Para ello debemos prestar atención a un método en cuestión, **navidagor.credentials.create()**.

Para un mejor entendimiento de este proceso se ha decidido que la mejor manera será visualizándolo tanto en un esquema como con el propio código. Por lo tanto, en la siguiente figura se muestra el flujo de mensajes típico que se dan durante el registro:

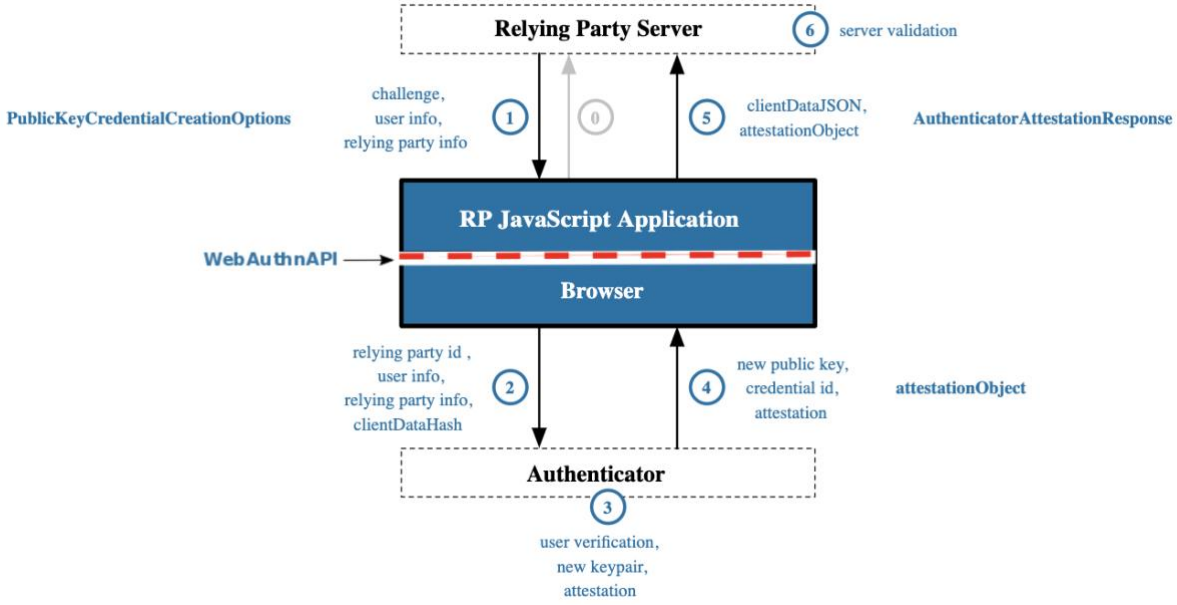


Ilustración 4 - Flujo de mensajes en el Registro (obtenida de [8])

De igual manera, podemos suponer que para que el flujo de comunicaciones se lleve a cabo, necesitaremos hacer uso de la API de WebAuthn, por ello a continuación se muestra un ejemplo básico de la utilización de la API durante un supuesto registro. Además, comentaremos cuales son los parámetros requeridos y que función realizan.

```
var challenge = new Uint8Array(32);
window.crypto.getRandomValues(challenge);

var userID = '11111'
var id = Uint8Array.from(window.atob(userID), c => c.charCodeAt(0))

var publicKey = {
    'challenge': challenge,

    'rp': {
        'name': 'Example Inc.'
    },

    'user': {
        'id': id,
        'name': 'alberto@example.com',
        'displayName': 'Alberto Ruiz'
    },

    'pubKeyCredParams': [{
        'type': 'public-key',
        'alg': -7
    }, {
        'type': 'public-key',
        'alg': -257
    }]
}

navigator.credentials.create({
        'publicKey': publicKey
    })
    .then((newCredentialInfo) => {
        console.log('SUCCESS', newCredentialInfo)
    })
    .catch((error) => {
        console.log('FAIL', error)
    })
```

Ilustración 5 - Ejemplo de navigator.credentials.create()

Al ser un ejemplo de prueba, en este caso estaremos simulando la comunicación con el servidor, ya que será él quien nos deba proporcionar la información necesaria para enviar al autenticador.
Así pues, comentaremos a continuación de forma ordenada aquellos atributos que son utilizados, relacionándolos a su vez con el flujo de comunicaciones anteriormente visto (Ilustración 4).

- **challenge**: Tal y como vemos en la primera comunicación de la Ilustración 4, una vez el usuario indica que quiere registrar un autenticador, el servidor (parte que confía) le responde enviándole diversa información, entre ellos una cadena aleatoría llamada ‘challengue’, que servirá para prevenir los ataques MITM.

- **rp:** Entre la información recibida del servidor también estará el objeto rp, básicamente en él se almacena información (nombre, icono, id) relativa al origen (aplicación web), que será enlazada a un par de claves y limitaremos así su alcance.

- **user:** Además, en las versiones más modernas de los autenticadores de FIDO (FIDO2, no en U2F), también es posible asociar el par de claves con información relativa a la cuenta de usuario, como por ejemplo su username o su id (se recomiendo no contener información personal).

- **pubKeyCredParams:** Para la generación del par de claves, el servidor le especifica una lista de algoritmos que soporta, para ello utiliza su identificador CBOR [11] mediante el atributo *alg*. Entre los algoritmos utilizados podemos destacar RS256, PS256 o el basado en curvas elípticas (ECDSA) ES256.

Además, aparte de los parámetros anteriormente vistos podemos indicarle otros con distintas finalidades:

- **timeout:** Podemos indicarle un tiempo máximo para que el usuario responda a utilizar el autenticador.

- **excludeCredentials:** Con el fin de prevenir el registro de más de par de credenciales para el mismo fin (por ejemplo, un usuario ya registrado quiere volver a registrar otra credencial en el mismo autenticador), el servidor puede especificar una lista de credenciales, donde el autenticador tendrá que comprobar que no posee ninguna de ellas, ya que de ser así se cancelará la operación.

- **authenticatorSelection:** Con este objeto especificaremos preferencias sobre el autenticador. Entre sus atributos podremos encontrar 'authenticatorAttachment', donde indicaremos que tipo de autenticador queremos que se utilice, 'platform' para el caso de autenticadores integrados o "cross-platform" si deseamos autenticadores de itinerancia (tokens físicos externos).
  Otro de sus atributos relevantes y que da muchas posibilidades de uso es el de 'requireResidentKey', con él podremos especificar que el tipo de credencial a crear sea del tipo residente. Este tipo de credenciales posteriormente (en la autenticación) podrán ser utilizadas y accesibles tan solo indicando el RP ID del servidor, donde el usuario podrá escoger (dentro de una lista de claves) la credencial a utilizar, útil para autenticaciones de factor único sin nombres de usuario ni contraseñas.
  Por último, tal y como comentamos en anteriores ocasiones, hay casos en los que se pide que el usuario interactúe con el autenticador a través de un gesto (biométrico, pin, botón, etc). Por lo tanto, el servidor podrá indicar si se requiere o no un gesto con el atributo 'userVerification'. Si este tiene el valor de 'required', se necesitará un gesto obligatoriamente (si el autenticador no tiene la posibilidad, saldrá un error), mientras que con el valor 'preferred' se impondrá un gesto solo si el autenticador posee esa posibilidad. Lógicamente también podremos eliminar la necesidad de un gesto del usuario, con el valor 'discouraged'.

- **attestation:** El servidor podrá especificar el nivel de confianza para la 'attestation', ya que por ejemplo si especificamos "direct", estaremos diciéndole al autenticador que nos muestre y certifique información relativa sobre él (certificados de fábrica, AAGUID, etc) con tal de verificar su procedencia. También podremos indicarle "indirect" si queremos que el cliente anonimice la verificación o "none" (por defecto) si no necesitamos verificar la atestación del autenticador.

Una vez se tenga generado el objeto ‘publicKey’ con los parámetros necesarios, podremos realizar la llamada correspondiente a la API de WebAuthn (navigator.credentials.create()) para que esta se encargue de comunicarse con el autenticador e indicarle las acciones a ejecutar.

En este caso, el autenticador se encargará de recibir las opciones y generar un nuevo par de credenciales (tras pasar una serie de requisitor previos) asociado a la cuenta del usuario en la aplicación web indicada.
Posteriormente, el autenticador enviará de vuelta al cliente una serie de objetos:

```
PublicKeyCredential {
    id: 'CredentialID',
    rawId: ArrayBuffer(59),
    response: AuthenticatorAttestationResponse {
        clientDataJSON: ArrayBuffer(121),
        attestationObject: ArrayBuffer(306),
    },
    type: 'public-key'
}
```

Ilustración 6 - Datos recibidos por el autenticador durante el registro

- **ID:** Corresponde con el valor único que referencia al nuevo par de claves generado. Deberá ser almacenado y usado por el servidor para que posteriormente pueda identificarlas.

- **clientDataJSON:** Este objeto aloja la información que el cliente le trasmitió al autenticador en un principio (challenge, origin, type).

- **attestationObject**: Este objeto, quizás el de más importancia en este proceso, según el nivel de confianza indicado por el servidor (tipo de ‘attestation’) revelará su procedencia y capacidades (certificados y firmas de certificación, formato, etc). Además de otra información sobre los datos del par de claves generadas (clave pública).

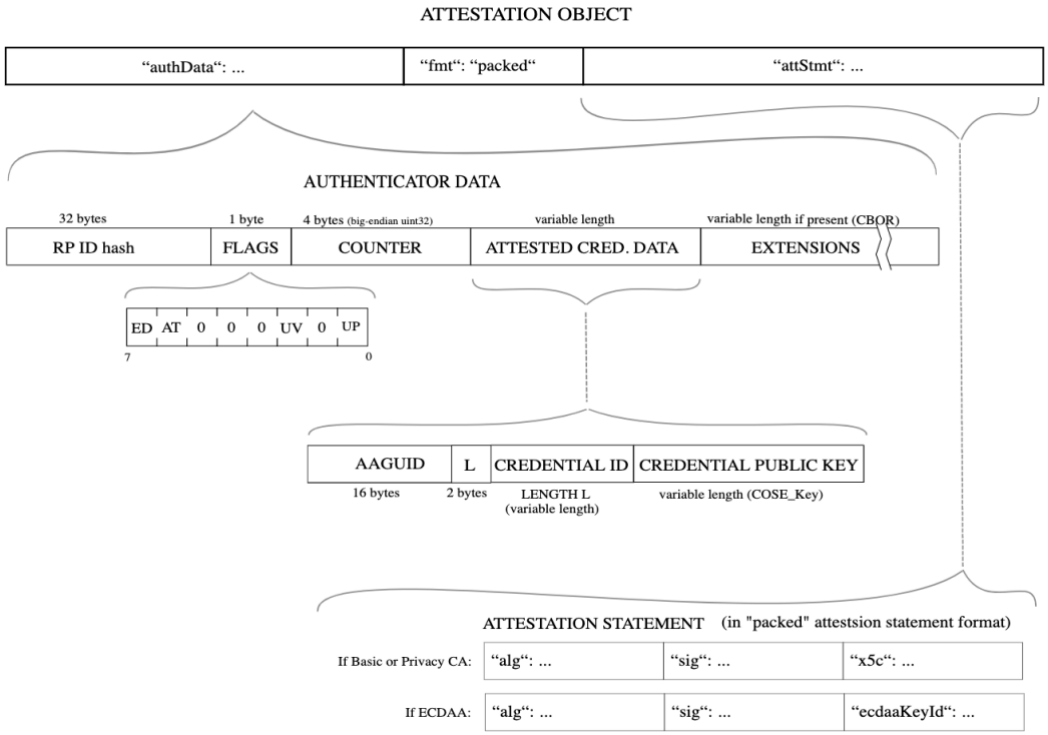


Ilustración 7 - Estructura del objeto 'attestationObject' (obtenida de [8])

- **authData:** AuthenticatorData, contiene los datos sobre el registro del par de claves, especialmente la clave pública y su identificador ID, además de otra información, como los flags especificados o el contador de firmas.

- **fmt:** Revela el formato de la ‘attestation’, útil para indicarle al servidor la forma en que deberá parsear y validar los datos.

- **attStmt:** También catalogado como ‘attestation statement’, variará dependiendo el formato de attestation (fmt), el objetivo principal de esta información es el de proporcionarle al servidor una manera para verificar la procedencia y habilidades del autenticador (proporcionando certificados, firmas, etc).

Por último, dichos objetos serán devueltos al servidor de la plataforma, el cual tendrá la responsabilidad de verificar y almacenar información relevante sobre el par de claves generado del usuario, para que posteriormente puedan ser utilizadas correctamente durante el proceso de autenticación.

3.3.2 Autenticación

Si en el anterior caso hablábamos sobre el proceso de registro y generación de credenciales, en este apartado detallaremos información relevante sobre el proceso donde se utilizan las claves previamente creadas, el de autenticación.

En este caso deberemos de prestar atención a otro método de la API, el de **navidator.credentials.get()**.

De igual manera que se hizo con el registro, a continuación se mostrará una ilustración donde podremos visualizar de forma correcta el flujo de mensajes durante el proceso de autenticación:

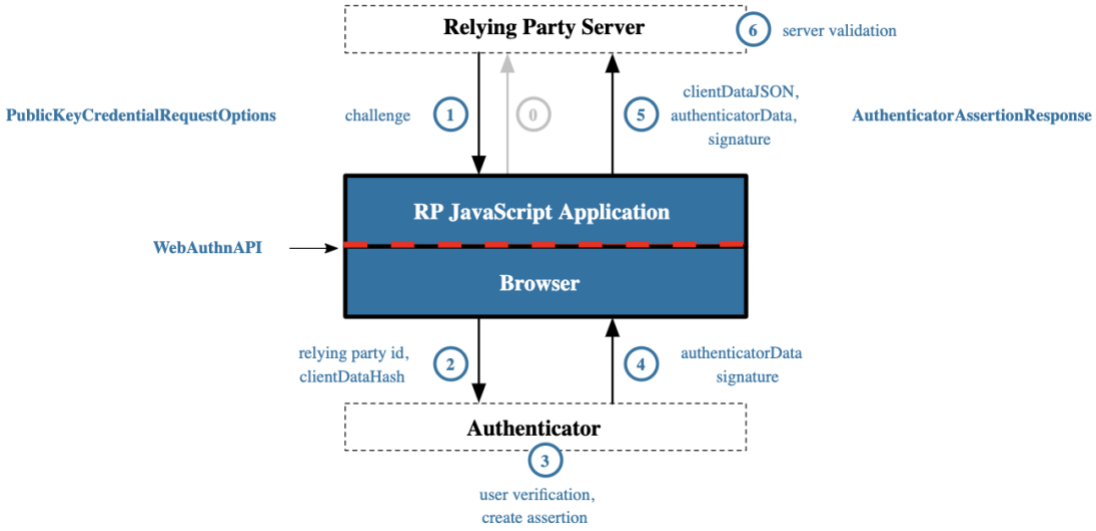


Ilustración 8 - Flujo de mensajes en la Autenticación (obtenida de [8])

A su vez, se ha incluido otra ilustración donde se refleja un código de ejemplo que hace uso de la API de WebAuthn para la autenticación. Además, comentaremos la definición de los parámetros y atributos más relevantes.

```
var publicKey = {
    challenge: challenge,

    allowCredentials: [
        { type: "public-key", id: credentialId }
    ]
}

navigator.credentials.get({ 'publicKey': publicKey })
  .then((getAssertionResponse) => {
      alert('Assertion recibida')
      console.log(getAssertionResponse)
  })
  .catch((error) => {
      alert('Assertion fallida')
      console.log(error)
  })
```

Ilustración 9 - Ejemplo de navigator.credentials.get()

Como ya se dijo de igual manera antes, estamos ante un entorno simulado, por lo que las peticiones/respuestas con el servidor y autenticador no serán reales. A continuación detallaremos la definición y utilidad de aquellos parámetros que podrán/deberán ser incluidos en el objeto 'publicKey', el cual enviaremos al autenticador por medio del método navigator.credentials.get():

- **challenge:** Cadena aleatoria generada y enviada desde el servidor (mensaje 1 de la ilustración 6) y que el autenticador deberá firmar (junto a otros datos) con la clave privada que considere oportuna.

- **allowCredentials:** El servidor podrá (ya que no es obligatorio) una lista de IDs de credenciales que el autenticador podrá usar (revisará entre sus claves para buscar coincidencia) para generar la 'assertion' del desafío.

Así pues, una vez el objeto haya sido transferido al autenticador por medio de WebAuthn, el dispositivo responderá con lo que denominamos un Assertion. En dicha respuesta, al igual que pasaba en el AttestationObject del registro, incluirá una serie de información:

```
PublicKeyCredential {
    id: 'CredentialID',
    rawId: ArrayBuffer(59),
    response: AuthenticatorAssertionResponse {
        authenticatorData: ArrayBuffer(191),
        clientDataJSON: ArrayBuffer(118),
        signature: ArrayBuffer(70),
        userHandle: ArrayBuffer(10),
    },
    type: 'public-key'
}
```

Ilustración 10 - Datos recibidos por el autenticador durante la autenticación

- **authenticatorData:** Similar al objeto authData del Attestation de registro, pero con la notable diferencia de que en este no se incluye la clave pública utilizada.

- **clientDataJSON:** Al igual que en el registro, es el conjunto de datos que el navegador le transmitió al autenticador (origin, type, challenge).

- **signature:** Uno de los objetos más relevantes, ya que es la firma generada con la clave privada sobre la concatenación de la información del autenticador y la recibida desde el servidor (hash de los datos del cliente).

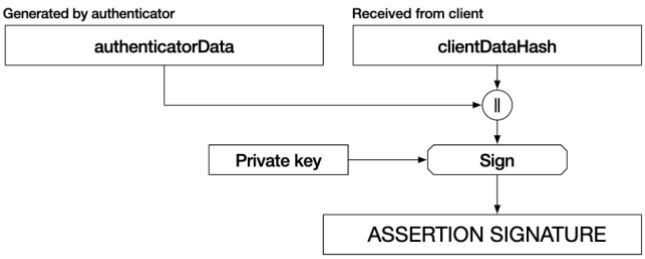


Ilustración 11 - Generación de signature en un Assertion

- **userHandle:** Contiene el valor del user.id que se definió cuando se generó la credencial (en el registro). Tal y como se dijo en el apartado de la terminología de WebAuthn, este valor es muy útil cuando se desea hacer uso de las claves residentes (RK), donde podremos también auntenticarnos sin usuario ni contraseña.

Además, en dicha respuesta se incluirá más información, como por ejemplo el **ID** de la credencial utilizada (útil para buscar su respectiva clave pública en el lado del servidor) y su tipo.

Posteriormente en el servidor, se deberá verificar la signature con la respectiva clave pública del usuario en cuestión (entre otros filtros previos). Si dicha comprobación coincide con el desafío que se envió en un principio, dará como válida la autenticación.

Tal y como hemos podido observar, la información que se envían entre ambos puntos no es secreta (no se usan ni envían contraseñas), por lo que mantiene este método seguro ante las amenazas que existían con los otros modos tradicionales.

Así pues, ya comentadas las funcionalidades principales de la API WebAuthn en el dispositivo del cliente, hablaremos en los próximos capítulos sobre los distintos puntos de vista también relacionados, como son el servidor (parte que confía) y el autenticador.

3.4 Modelo autenticador WebAuthn

Tal y como hemos nombrado en multitud de ocasiones, en este método de autenticación es necesaria la utilización de tokens físicos de seguridad, donde se realicen todos los procesos criptográficos y almacenen las respectivas claves (entre otra información).

La API de WebAuthn implica un modelo funcional abstracto para el autenticador, es decir, las plataformas cliente pueden implementar ese modelo de la manera que deseen, pero el comportamiento de la API deberá ser indistinguible, independiente del modelo del autenticador.

Desde el punto de vista de los autenticadores, este modelo define aquellas operaciones y formatos que deben poder admitir. De otro modo, este modelo

no define la forma en que han de comunicarse con el cliente, por lo que para WebAuthn será indiferente si el dispositivo utiliza un medio de transporte NFC, USB o BLE.
Aún así, como se nombró en el capítulo anterior, existen diversos parámetros tanto en el registro como en la autenticación, donde los servidores (partes que confían) pueden influir sobre los autenticadores, como por ejemplo especificar el tipo de dispositivo, necesidad de gestos de usuarios, etc.

En este momento, podremos nombrar una serie de características que consideramos importantes sobre los autenticadores y su funcionamiento:

- En el autenticador, ya sea en dispositivo interno o integrado, proporciona una administración de claves y firmas criptográficas (generación y uso).

- En cada autenticador existirá un mapa de credenciales, donde se almacenen los pares de claves generados y que serán usados para construtir ‘assertions’ durante la autenticación.

- Cada autenticador tiene un valor de AAGUID, una cadena de una longitud de 128 bits que identifica el tipo de autenticador (por ejemplo su marca y modelo). En este momento entra en juego el servidor, ya que este podrá influir sobre el nivel de confianza que posee sobre determinados autenticadores, por ello podrá exigir diferentes tipos de ‘attestation’ para certificar su procedencia y habilidades.

- El esquema de generación de firmas debe ser lo más eficiente posible, sobretodo en casos donde el enlace entre el dispositivo cliente y el autenticador tiene un ancho de banda o latencia limitado.

- Los datos que procesan los autenticadores deberán ser pequeños y fáciles de entender desde un bajo nivel (no deberían interpretar codificaciones como JSON).

- Cliente y autenticadores deberán tener la posibilidad de agregar enlaces contextuales dependiendo del caso (nivel de confianza del servidor sobre el autenticador).

- Los autenticadores deberían implementar un contador de firmas, de tal forma que se incremente con cada autenticación de éxito realizada (assertion). Esto ayudará a los servidores a detectar la existencia de autenticadores clonados, ya que el valor del contador no sería el esperado.

- Todos los autenticadores dentro del ámbito de autenticación tienen la característica de “algo que se tiene”, pero además podrán implementar mecanismos para verificar “algo que se es” (biometría) o “algo que sabe” (PIN) sobre el usuario legítimo, por lo que podrán ser compatibles con métodos de autenticación de múltiples factores. Como ya se dijo, el servidor (parte que confía) podrá exigir la utilización de estos métodos.

- Deberán proporcionar alguna forma de certificación a las partes que confían en WebAuthn. Con esto nos referimos al tipo de 'attestation' requerida por el servidor con el fin de determinar el nivel de confianza sobre dicho autenticador (veremos en el apartado 3.6 el formato de un 'attestation').

Así pues y aunque ya podríamos imaginarlo, existen diversos casos en los que el autenticador generará firmas criptográficas:

- Durante el registro de una nueva credencial, el autenticador genera una **firma de certificación**, donde proporciona al servidor una prueba sobre su procedencia y capacidades. Este tipo de firmas puede testificar por ejemplo su AAGUID.

- En cambio, durante la etapa de autenticación se genera una firma que tiene como objetivo afirmar que el autenticador posee el par de claves correspondientes y que el usuario da su consentimiento (**firma de 'assertion'**). La firma se realiza con la clave privada sobre la concatenación de los datos del autenticador y del hash de los datos del cliente, tal y como podemos visualizar en el campo 'signature' (Ilustración 8).

Un ejemplo de implementación de dicho modelo es la detallada por FIDO Client to Authenticator Protocol (CTAP) [7]. Este protocolo como dijimos en líneas anteriores, permite que dispositivos externos (como teléfonos móviles o tokens de seguridad FIDO) funcionen con la API de WebAuthn y sirvan como autenticadores, en nuestro caso especialmente para servicios web.

En este caso podríamos hablar de FIDO2, una nueva especificación de autenticador que hace uso del protocolo CTAP2.

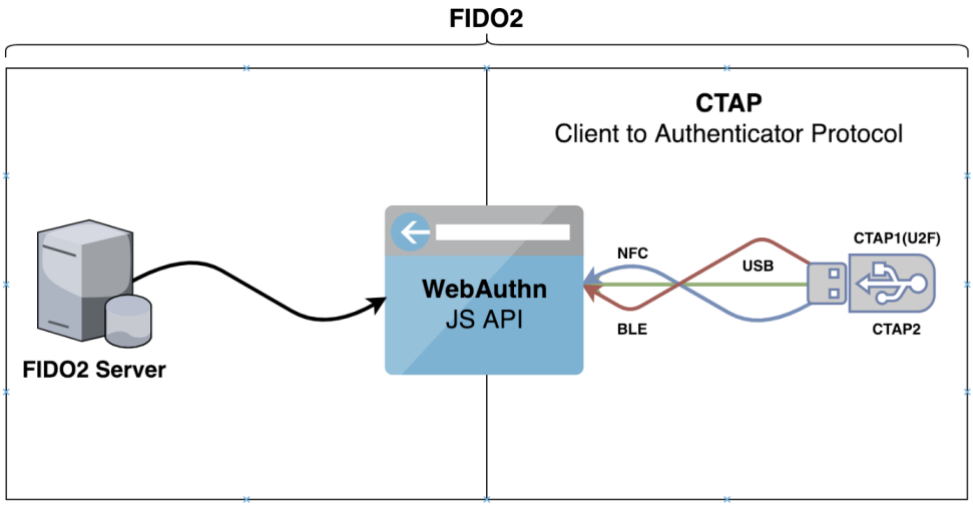


Ilustración 12 - Componentes FIDO2 (obtenida de [10])

FIDO2 es el sucesor del protocolo FIDO Universal 2nd Factor (U2F), tiene todas las ventajas que su antecesor poseía, pero con la principal diferencia de que un autenticador FIDO2 es multifactor (algo que tiene).

Este tipo de autenticadores, tal y como hemos podido comentar en ocasiones, pueden funcionar tanto en modo factor único (por ejemplo se necesita la acción del usuario como prueba de presencia, una simple pulsación de un botón) como en multifactor (el autenticador verificará al usuario mediante pruebas como un PIN o datos biométricos).
Aún así, en cualquier caso estamos ante un autenticador y método de autenticación que no utiliza contraseñas, la característica principal de WebAuthn.

Por último, cabe destacar que a pesar de que WebAuthn está enfocado hacia FIDO2, sigue teniendo compatibilidad con autenticadores que funcionen mediante U2F (y por lo tanto CTAP).

3.5 Operaciones en el servidor que confia en WebAuthn

Como hemos podido ver en imágenes anteriores (por ejemplo la ilustración 4), tanto en el proceso de registro como en el de autenticación intervienen una serie de componentes, un autenticador, el cliente (WebAuthn API) y la parte que confía (servidor).
Durante este apartado detallaremos aquellas acciones que se deberían realizar desde el punto de vista del servidor, ya que a pesar de no tomar una atención relevante dentro de la especificación de WebAuthn, sus funcionalidades serán igual de importantes en el método de autenticación, verificando cada información recibida para registrar/autenticar a un usuario dado.

Así pues, de igual manera que en la anterior sección, comentaremos las acciones desde un punto de vista teórico, ya que en capítulos posteriores (4) podremos reflejarlo en un supuesto práctico.

3.5.1 Registro de una nueva credencial

Como ya se reflejó en el apartado 3.3.1 Registro, una vez el autenticador ha registrado un nuevo par de credenciales, éste retornaba al cliente un objeto del tipo ‘AuthenticatorAttestationResponse’, donde contiene objetos cuya información referencia a los datos del cliente (clientDataJSON), al autenticador y al par de claves generadas (attestationObject).

Así pues, dicha información deberá ser enviada al servidor (parte que confía), para que la verifique y realice las acciones oportunas sobre el usuario:

1. En primer lugar, se verificará que el objeto clientDataJSON recibido sea del tipo JSON, válido para su correcta interpretación.

2. Comprobar que el valor de clientDataJSON.type es “webauthn.create”.

3. Verificar que el valor de clientDataJSON.challenge es el mismo que se envió al cliente antes de la creación del par de credenciales.

4. Verificar que el valor de clientDataJSON.origin corresponde con el servidor.

5. Calcular el hash SHA-256 del objeto clientDataJSON.

6. Realizar la decodificación CBOR del objeto attestationObject, para obtener los valores de declaración de certificación *fmt*, los datos de autenticación *authData* y la declaración de certificación *attStmt*.

7. Verificar que el rpidHash del objeto authData corresponda con el hash-256 del RP ID esperado por el servidor.

8. Si se requiere la presencia del usuario para el registro, comprobar que el bit UP de *flags* está activado.

9. Si se requiere verificación del usuario para el registro, comprobar que el bit UV de *flags* está activado.

10. Verificar que los valores de *extensions* en authData corresponde con los esperados.

11. Determinar el formato de la declaración de certificación del valor de *fmt*, verificando que está entre el conjunto de valores admitidos.

12. Comprobar que el valor de *attStmt* es una correcta declaración de certificación, verificando la firma de certificación según el procedimiento de verificación de cada *fmt*, respecto a los valores de *attStmt*, *authData* y el hash del objeto clientDataJSON.

13. Si la verificación anterior es correcta, se deberá obtener una lista de certificados para este tipo de formato (por ejemplo el servicio de metadatos de FIDO proporciona información en base al AAGUID del autenticador).

14. Evaluar el nivel de confianza de la certificación en base a los resultados del procedimiento de verificación anterior:

    a. Si el autenticador utilizó una auto-certificación, se comprobará que es válido en base a la política del servidor (si permite o no los autofirmados).
    b. Si uso ECDAA, se verifica que el identificador de clave pública esté incluido dentro de la lista de certificados anteriormente obtenida (de confianza).
    c. De lo contrario, se deberá verificar haciendo uso de los certificados X.509 recibidos (en los objetos) por el procedimiento de verificación, que encadena correctamente a un certificado de raíz aceptable.

15. Verificar que el ID de la credencial no se ha registrado previamente.

16. Por lo tanto, si la declaración de certificado *attStmt* fue exitosa y obtuvo un nivel de confianza aceptable, se aceptará el registro, asociando y almacenando la clave pública recibida junto a su ID con los datos del usuario en cuestión.

17. De otro modo, si el nivel de confianza no es de confianza se anulará el registro.

Aún así, si la política del servidor lo permite, se podrá registrar la credencial a pesar de no tener información suficiente para testificar que la procedencia de dicho par de claves ha sido generada por un modelo de autenticador en particular.

3.5.2 Verificar un 'assertion' de autenticación

De otra forma, cuando el usuario desea acceder a una plataforma e identificarse haciendo uso de un autenticador (búsqueda del par de claves y firma del desafío recibido), este retornará al cliente un objeto del tipo 'PublicKeyCredential', cuya respuesta 'assertion' contiene información necesaria para su posterior verificación (detallada en el apartado 3.3.2 Autenticación).

Por lo tanto, a continuación se especifica una serie de fases que el servidor debería procesar sobre dichos objetos:

1. En primer lugar, se verificará que el ID del par de claves corresponde entre los aportados por 'allowCredentials' (si se utilizó).

2. Verificar que la clave utilizada (a través de su ID) corresponde con el usuario que se está autenticando. Se podrá hacer uso del valor *userHandle* recibido para su búsqueda.

3. Mediante el ID de la credencial, se buscará en el servidor la clave pública correspondiente.

4. Concatenar los valores de clientDataJSON, authenticatorData y signature.

5. Verificar que clientDataJSON tiene un formato correcto para su interpretación.

6. Verificamos el valor de clientDataJSON.type sea igual a "webauthn.get", que corresponde a la acción de autenticar un desafío/usuario.

7. Verificamos que el valor de *challenge* es el mismo por el esperado (el que el servidor envió en un principio).

8. Verificar que el valor de *origin* corresponde con la dirección del servidor (parte que confía).

9. Comprobar que el valor de authenticatorData.rpIdHash sea el mismo que el hash SHA-256 del RP ID.

10. Si se requiere la presencia del usuario para la autenticación, comprobar que el bit UP de *flags* está activado.

11. Si se requiere verificación del usuario para la autenticación, comprobar que el bit UV de *flags* está activado.

12. Verificar que los valores de *extensions* en authData corresponde con los esperados.

13. Calcular el hash SHA-256 sobre el objeto clientDataJSON (datos del cliente).

14. Usando la clave pública recogida en el paso 3, se verificará que el valor de *signature* corresponde con la concatenación de authenticatorData y el hash de clientDataJSON.

15. Comprobar el valor del contador de firma comparándolo con el valor almacenado en el servidor. Como ya se comentó anteriormente, un valor inesperado podría reflejar que existe una duplicación del autenticador (existen al menos dos copias de la misma clave).

16. Si todos los pasos anteriores son correctos, se podrá dar por autenticado al usuario.

Así pues, dependiendo de los resultados obtenidos durante esta ceremonia, la plataforma web decidirá que acciones realizar (denegar el acceso en caso erróneo o proporcionárselo en caso de éxito).

3.6 Posibles escenarios

Una vez entendido el funcionamiento de WebAuthn y las responsabilidades que tiene cada componente (servidor, cliente y autenticador), me parece oportuno comentar diversas situaciones que se pueden dar con la utilización del estándar.
Así pues, en esta sección se detallará de forma breve y teórica los aspectos más importantes a tener en cuenta en cada uno de los siguientes escenarios.

3.6.1 Utilización de WebAuthn como segundo factor de autenticación (2FA)

Este sería el caso de escenario más simple, donde el usuario utilizará el autenticador como segundo factor de autenticación, esto significa que tanto el nombre de usuario como su respectiva contraseña se seguirán usando.

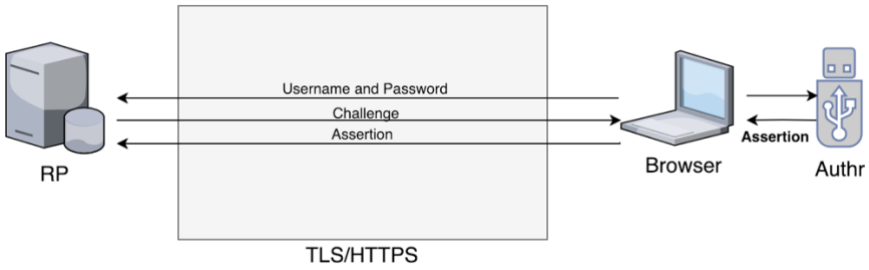


Ilustración 13 - WebAuthn como segundo factor de autenticación (2FA) (obtenida de [10])

Lo negativo de utilizar este método como segundo factor es que todavía se seguirán usando contraseñas, enviando así información secreta a través de la red. Así pues, este tipo podría asemejarse a los métodos comentados en el capítulo 2 (Estado del arte), donde se nombraban aquellos como códigos temporales TOTP o de un solo uso a través de sms, email, etc.
Respecto a su funcionamiento, el servidor deberá verificar en primer lugar los valores del usuario y contraseña, pasando al segundo método de autenticación (WebAuthn) en caso de éxito. Además, el servidor podrá referenciar a las claves del usuario por medio del campo 'allowCredentials'.

*Nota: Cabe destacar que la utilización de WebAuthn solo es posible en aquellas plataformas web que utilicen un canal de comunicación seguro (TLS/HTTPS), por lo que en los posteriores escenarios se dará por hecho este aspecto.

3.6.2 Utilización de WebAuthn sin envío de contraseñas

En este segundo escenario nos introducimos en una de las características importantes de la utilización de WebAuthn, y es que la notable diferencia respecto al escenario anterior es que ahora no se utilizan las contraseñas, es decir, no se envía información secreta por la red.

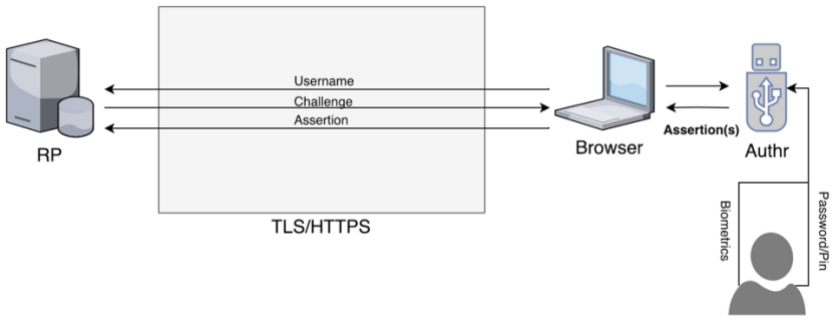


Ilustración 14 - Utilización de WebAuthn sin contraseña (obtenida de [10])

Sin embargo, para que el servidor pueda confiar en que efectivamente el usuario que utiliza el autenticador es el legítimo, es aconsejable requerir una verificación por parte de dicho usuario, detallándolo a través del parámetro 'userVerification' con el valor de 'required' durante el registro y autenticación.

Esto hará que el usuario tenga que autenticarse sobre el dispositivo, como por ejemplo mediante un PIN o datos biométricos.

Así pues, estamos ante un gran avance en cuanto a los métodos de autenticación nos referimos, disminuyendo las probabilidades de amenazas de manera considerable.

3.6.3 Utilización de WebAuthn sin usuarios ni contraseñas

Un paso más adelante sobre el escenario anterior, sería un escenario en el que no sea necesario la utilización de nombres de usuario ni contraseña, tan solo el autenticador.
Esta configuración es posible gracias a las claves con la propiedad de residentes (RK), éstas, como ya se comentó en la terminología, es un tipo especial de claves que se podrán utilizar durante la autenticación sin el conocimiento previo de ellas. Para especificarlo, durante el registro de la credencial se deberá detallar el valor de 'requireResidentKey' a 'true' (si el autenticador no lo soporta, saldrá un error).

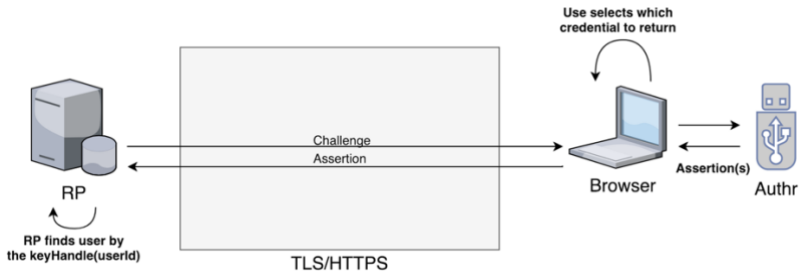


Ilustración 15 - Utilización de WebAuthn sin usuario ni contraseña (obtenida de [10])

Así pues, durante la autenticación el servidor tan solo le proporcionará un challenge (prescindimos de 'allowCredentials'). Esto hará que el autenticador le muestre al cliente (y por tanto al usuario) una lista de credenciales residentes que podrían satisfacer al RP en cuestión, donde el usuario deberá elegir una de ellas con la que generar el 'assertion'.
Llegados a este punto podríamos pensar, si no se envía ninguna información referente al usuario, ¿cómo hace el servidor para identificar al usuario?, pues gracias al valor 'userHandle' alojado en el objeto 'assertion' devuelto por el autenticador. Este valor corresponde con el 'user.id' que se utilizó durante el registro, por lo que se podrá asociar al par de claves correspondiente, para verificar si la autenticación es o no correcta.

Cabe destacar que las claves residentes solo están disponibles en autenticadores FIDO2, no en los antiguos U2F. Además, actualmente solo el navegador Edge de Windows puede realizar este procedimiento con este tipo de claves.

3.6.4 El usuario reporta la perdida de la credencial

En este caso, si el usuario reporta una perdida de la credencial/autenticador, el servidor debería proporcionarle una lista de las credenciales registradas, seleccionando así aquella que se haya perdido para eliminarla de la base de datos.
Así, si posteriormente el servidor recibe un assertion con dicha credencial, lo rechazará ya que no está entre las permitidas por el usuario en cuestión.

3.6.5 Debido a la inactividad, el servidor anula una credencial

De igual forma que se hacía en el anterior escenario, si el servidor considera que se deba anular una credencial por su inactividad, será eliminada de la base de datos, por lo que en posteriores autenticaciones no serán incluidas en la lista de 'allowCredentials'. Si alguna 'assertion' se firmara con dicha credencial, sería igualmente rechazada.

3.6.6 El usuario elimina la credencial del autenticador

En este caso, será en el propio autenticador donde el usuario decida eliminar una credencial (en el servidor seguirá estando almacenada). En posteriores autenticaciones no se podrá utilizar el par de claves (ya que serán rechazadas por el autenticador).
Por lo tanto, será el propio usuario el encargado de eliminarlas de su base de datos debido a una inactividad.

3.7 Ventajas y desventajas

Así pues, como ya hemos podido apreciar en los apartados anteriores, se han proporcionado las características y aspectos fundamentales para entender el estándar WebAuthn y comprender las finalidades y límites existentes para su uso en el ámbito web.

Por lo tanto, a continuación incluiremos una serie de conclusiones que hemos detectado, tanto las ventajas como desventajas que se obtendrán al implantar y utilizar WebAuthn:

3.7.1 Ventajas

- La utilización de contraseñas débiles (depende de la longitud, significado, caracteres especiales, números, mayúsculas y minúsculas) o la reutilización de las mismas pasa a un segundo plano, ya que la importancia recae sobre el método de autenticación (ya sea de primer o segundo factor) que hace uso de WebAuthn.

- Con WebAuthn, los usuarios no tienen porqué depositar confianza sobre el servidor, ya que no nos debería de preocupar la forma en que administra la información relativa al estándar.

- Tal y como comentamos en anteriores ocasiones, con WebAuthn eliminamos multitud de amenazas que existen sobre los otros métodos de autenticación, como por ejemplo phishing, MITM, etc.

- De forma conjunta a la anterior ventaja, al no hacer uso de contraseñas los servidores no tendrán amenazas sobre robos o filtraciones de credenciales en sus bases de datos.

- De cara a los desarrolladores, WebAuthn implementa un modo de autenticación para entornos web, lo que supone una liberación de responsabilidad de diseñar uno nuevo para incorporarlo.

- Las plataformas no tienen porque proporcionar hardware a sus usuarios, sino que estos pueden utilizar cualquier autenticador (según la política de confianza), además de que en un mismo dispositivo es posible almacenar multitud de credenciales.

- Por último, cabe destacar la aceptabilidad por parte de los navegadores.

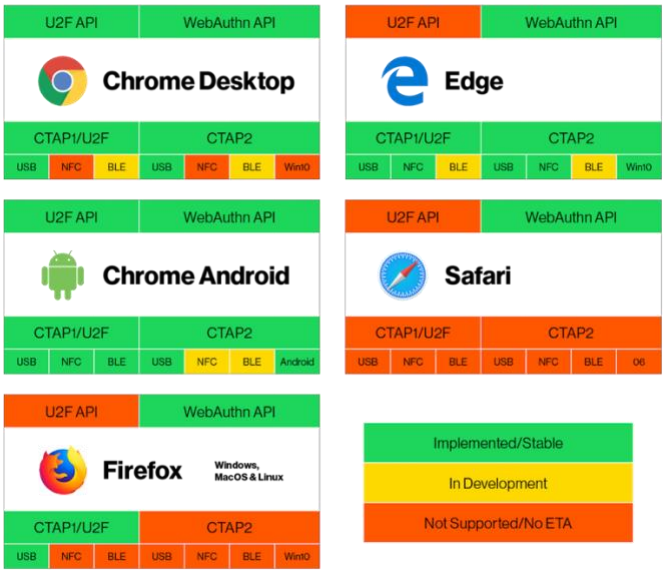


Ilustración 16 - Compatibilidad de los navegadores con WebAuthn

Tal y como vemos en la ilustración anterior, actualmente WebAuthn es compatible con la mayoría de los navegadores, a pesar de todavía quedan protocolos por implementar. Se espera que con el tiempo el estándar WebAuthn esté totalmente integrado (ver recurso [17] para información actualizada).

3.7.2 Desventajas

- En cuanto a las desventajas encontradas, desde mi punto de vista sobre la seguridad de WebAuthn diría que hasta el momento no tiene ninguna, ya que se podría decir que es uno de los métodos de autenticación más seguros en el ámbito web. Sin embargo, por decir alguna podría

nombrar la necesidad de usar un dispositivo hardware para autenticarse, esto podría resultar incomodo para los usuarios. Aún así, los beneficios que obtendríamos en comparación con las desventajas son considerables.

En conclusión, hemos podido definir y comprobar la manera en que el nuevo estándar WebAuthn tiende a convertirse en el método de registro/autenticación más seguro en servicios web. Así pues, está ahora en poder tanto de las plataformas web como de los navegadores, implantar e incentivar a los usuarios a que hagan uso de este método, hacer una web cada vez más segura y resistente ante posibles amenazas.

4. Ejemplo práctico: Incorporación de WebAuthn como método de autenticación

A lo largo de los anteriores capítulos hemos podido conocer todas aquellas características importantes que rodean al nuevo estándar WebAuthn, desde los motivos que han llevado a su creación hasta los aspectos técnicos de funcionamiento.
Como se ha ido recalcando durante el documento, este TFM tiene como objetivo principal dar a conocer dicho protocolo, apreciar los beneficios que podríamos obtener con su implantación (tanto desarrolladores como usuarios) e imaginar los posibles casos de uso que podrían existir.

Así pues, una vez hayamos comprendido las bases de WebAuthn, nos introducimos en este capítulo con la intención de mostrar un básico ejemplo práctico del estándar en una supuesta plataforma web. Se detallarán los aspectos más importantes en cada una de las partes que interactúan en el proceso (cliente y servidor), así como también las conclusiones que hayamos podido obtener tras su puesta en marcha.

4.1 Diseño

Como hemos nombrado en la parte técnica de WebAuthn (capítulo 3.3), su modo de funcionamiento se resume a las acciones de registro y autenticación de credenciales (con tokens de seguridad). De esta manera, se ha diseñado una estructura de elementos en los que poder reflejar las responsabilidades y decisiones que recaen entorno a esta especificación, es decir, se ha creado un ejemplo en el que se muestran tanto las acciones del servidor como del propio cliente que interactúa con WebAuthn.

Entrando en materia, se ha desarrollado una plataforma web en la que se mostrará el proceso de autenticación y registro, acompañado posteriormente de un panel de control donde se administrarán las credenciales asociadas a cada usuario. En este caso, se ha incluido WebAuthn como segundo factor de autenticación (2FA), siendo así necesario la utilización adicional de un usuario y su contraseña.
La idea general no es implantar WebAuthn en una página web real, sino realizar un ejemplo de un sistema de autenticación que haga uso del protocolo y que los usuarios puedan testear su modo de funcionamiento con sus tokens físicos de seguridad. Además, con la finalidad de facilitar su usabilidad, portabilidad e implantación, se han dockerizado los sistemas desarrollados.

Para poder visualizar de mejor manera la arquitectura, las comunicaciones que se llevan a cabo y los elementos que interactúan, se ha realizado dos ilustraciones que muestran ambos casos comentados, por un lado la acción de registro y por otro lado el de autenticación.

- **Registro**:

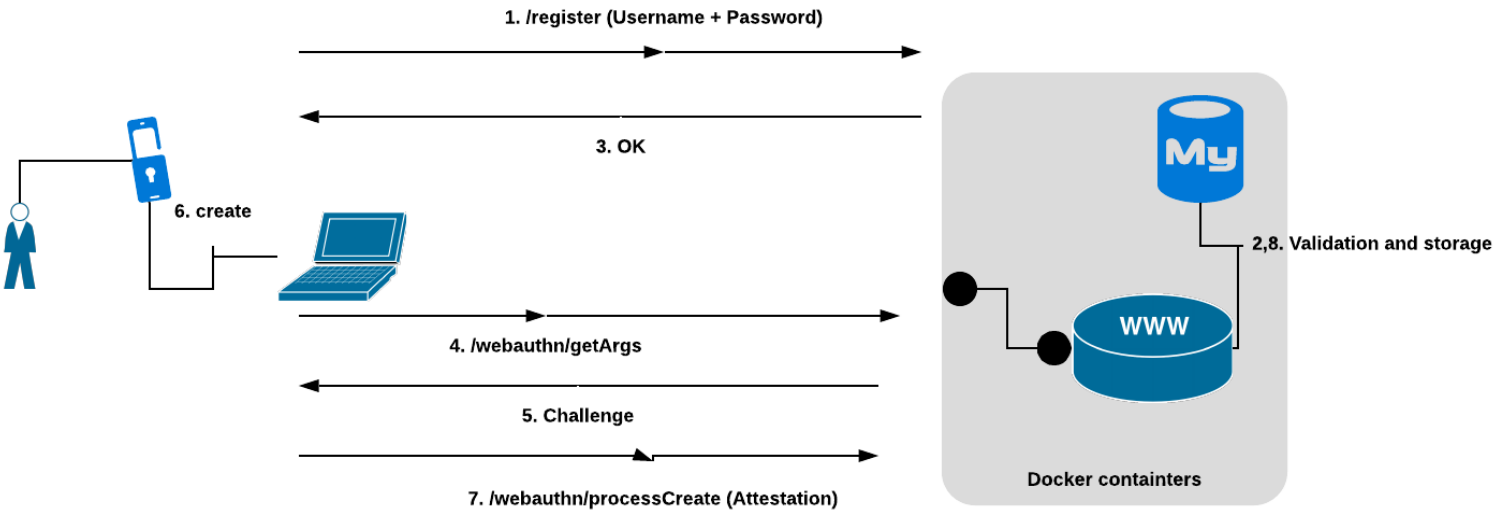


Ilustración 17 - Esquema durante la etapa de registro

- **Autenticación:**

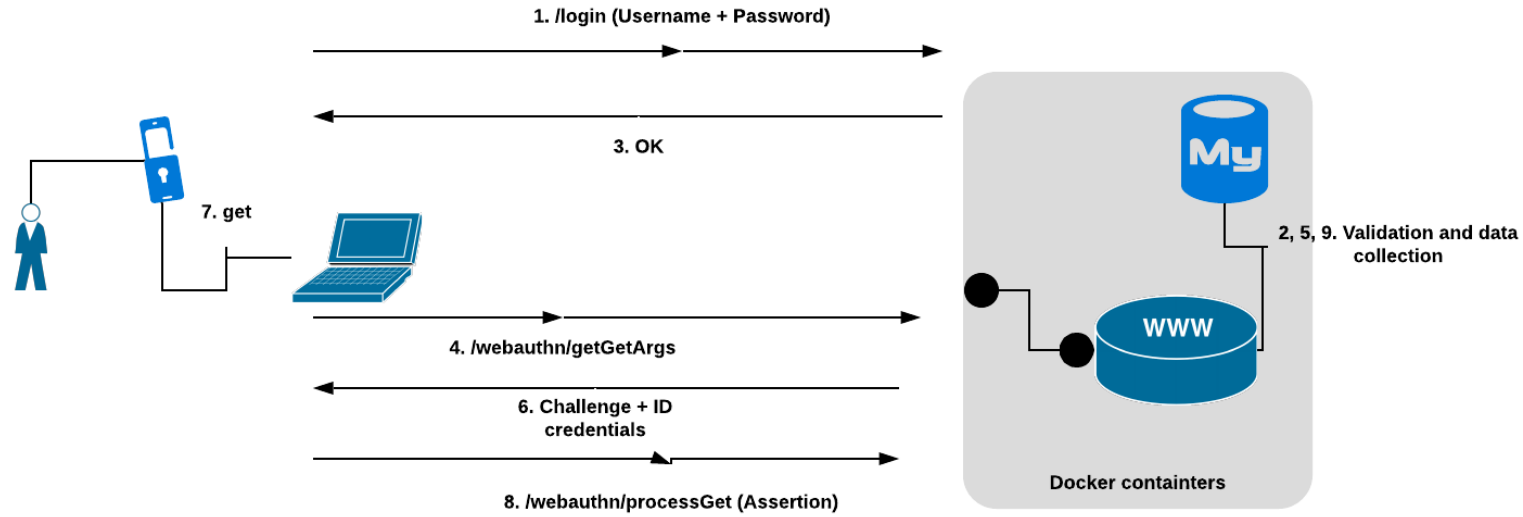


Ilustración 18 - Esquema durante la etapa de autenticación

Tal y como se ha podido visualizar, se ha detallado el orden de las comunicaciones y procesos que se llevan a cabo en ambas fases, además de las rutas que se utilizan para solicitar y enviar los datos correspondientes. El cliente, comandado por el navegador, el usuario y su autenticador, forma uno de los extremos que se comunicará con el servidor, el cual estará constituido por el propio servidor web, su base de datos y las librerías necesarias para procesar y validar los datos recibidos de los usuarios.

Así pues, ambas ilustraciones (17 y 18) serán continuamente referenciadas en los próximos apartados, donde detallaremos con más profundidad las características de cada elemento.

4.2 Cliente

Tal y como se indica en la terminología de WebAuthn (3.2) y en su documentación oficial [8], con el 'lado del cliente' nos referimos al conjunto de elementos formado por el usuario, el navegador (y por ende la máquina) y el autenticador (en nuestro caso un token físico de seguridad).

Refiriéndonos a la plataforma en cuestión, a pesar de que no haya una separación real entre cliente y servidor (las vistas se generan en el mismo servidor), identificaremos al cliente como la interfaz que visualiza el usuario, incluyendo además los scripts javascript necesarios para realizar las peticiones al servidor.
Así pues, haremos referencia a las ilustraciones 17 y 18 para explicar las diferentes acciones que se llevan a cabo desde el lado del cliente:

- **Registro**

  Como se dijo anteriormente, utilizaremos WebAuthn como segundo factor de autenticación, por ello previamente será necesario registrar una nueva credencial (usuario + contraseña), enviada a través de un formulario a la dirección /register de la plataforma.

  Una vez tengamos una cuenta de usuario, podremos ahora registrar claves asimétricas haciendo uso de tokens físicos.
  El procedimiento desde el punto de vista del cliente es bastante simple, en primer lugar se solicitará un desafío (entre otros datos) al servidor (a la dirección /webauthn/getArgs), pasando dicha información al método ya nombrado navigator.credentials.create(). Con dicha función, haremos que la API de WebAuthn realice todo el proceso de solicitar la interacción con un token para que se cree una nueva credencial en él. Si este proceso se lleva a cabo correctamente, el autenticador nos devuelve el objeto llamado Attestation, el cual entre otros datos contiene el ID de la credencial, su clave pública e información sobre el autenticador. Tan solo faltaría enviar dicha información al servidor para que este la procese, valide y almacene (a la dirección /webauthn/processCreate).

- **Autenticación**

  En cuanto a la autenticación, podríamos decir que el proceso es casi idéntico al de registro, donde varían los datos que se envían y las direcciones correspondientes.
  Por un lado, previamente será necesario poseer un nombre de usuario y una contraseña, que serán enviados a través del formulario a la dirección /login. Si esto se lleva a cabo y existe una clave asimétrica registrada, se solicitará su presencia. En primer lugar se obtendrá un desafío (incluyendo una lista de las credenciales aceptadas), solicitado a través de /webauthn/getGetArgs. Una vez se obtenga desde el servidor, se hará uso de la API de WebAuthn (navigator.credentials.get()) proporcionándole los datos necesarios. Esto hará que se solicite la

presencia del token que contiene una de las claves solicitadas, que en caso positivo generará el llamado Assertion, que entre otra información contiene el desafío firmado con la clave privada correspondiente.
Así pues, se enviará al servidor el resultado (dirección /webauthn/processGet), donde se deberá verificar que la firma se ha realizado con la clave solicitada, autenticando así al usuario en cuestión.

- **Eliminar credencial**

  Como se dijo en el apartado del diseño, la página web básicamente consiste en un panel de control en el que poder visualizar, crear y eliminar credenciales asimétricas asociadas al usuario. De esta manera, el usuario en cuestión podrá solicitar la eliminación de una determinada clave (a través de la dirección /deletecredential/{id}), por lo que ya no se solicitará su presencia (lista de IDs) en futuras autenticaciones.
  El motivo de su eliminación, tal y como se dijo en anteriores capítulos, podría ser el caso de pérdida del token físico, eliminando así también la posibilidad de que un tercero pueda utilizarla en el sistema.

Por último, me parece oportuno destacar que los datos devueltos por el autenticador vienen en formato buffer, por lo que tanto en el lado del cliente como en el del servidor será necesario tratar con conversiones en base64, lo que facilitará su transmisión y almacenamiento.

4.3 Servidor

Como es lógico, en la plataforma desarrollada casi toda la carga computacional recae sobre el servidor, ya que será el encargado de guiar al usuario hacia la autenticación, incluyendo así los sucesivos procesos que ocurren en su interior.

Tal y como se puede visualizar en las ilustraciones 17 y 18, el lado del servidor corresponde tanto al sistema de almacenamiento (base de datos relacional mysql) como al servidor web (apache). En este último se alojará la plataforma web desarrollada, la cual esta basada en el framework de php denominado Symfony (facilita la generación de vistas, controladores e inclusión de librerías). En cuanto a la base de datos, a continuación se puede visualizar las tablas de las que hace uso, ya que serán necesarias para almacenar la lógica que generan los usuarios (credenciales de usuarios y de tokens físicos):

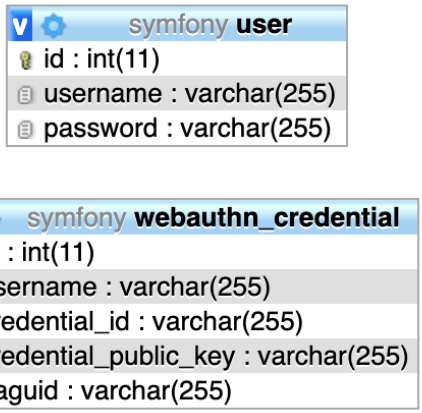


Ilustración 19 - Tablas de la base de datos

En este caso, al ser un ejemplo básico, tan solo será necesario almacenar el ID de la credencial y su respectiva clave pública. El motivo de esto viene dado por el tipo de Attestation que se requiere, ya que en la plataforma se ha configurado para que no se solicite (none) información adicional del autenticador (clave raíz, proveedor, etc), ya que así la página podrá ser testeada por cualquier token de seguridad, independientemente de la marca de procedencia.

En cuanto a la implementación del sistema, sin ninguna duda la popularidad recae sobre la librería que procesa los datos recibidos por WebAuthn. Al ser un estándar prácticamente reciente, todavía no existe gran variedad en cuanto a su implementación en varios lenguajes de programación. A pesar de ello, en nuestro caso utilizaremos una librería en php de un tercero que actualmente sigue siendo desarrollada [18], pero que aún así se adapta a nuestras necesidades.

Aunque en líneas posteriores comentaremos aspectos de ella, cabe destacar que es necesario poseer una instancia de openssl, ya que se hace uso de ella para verificar las autenticaciones de los usuarios.

De igual forma que comentábamos en el apartado anterior, para el caso del servidor realizaremos la misma estrategia, dividiremos su explicación en base a los diferentes procesos que se dan lugar:

- **Registro**

  En primer lugar, ya que el uso que le estamos dando a WebAuthn en esta plataforma es de segundo factor, para su correcto funcionamiento ha sido esencial la implicación de las sesiones (y sus respectivas cookies), manteniendo determinada información entre sucesivas peticiones cliente-servidor.

  De forma paralela al caso del cliente, cuando este nos solicita un nuevo desafío (/webauthn/getArgs), se le comunicará a nuestra librería la generación de un objeto JSON que incluye la información necesaria,

como por ejemplo datos que referencian al usuario, al desafío aleatorio, al tipo de attestation (none en nuestro caso) o a la verificación por parte del usuario.

Así pues, una vez que el autenticador haya registrado una nueva credencial y que el cliente nos devuelva dichos objetos (attestation) a la dirección /webauthn/processCreate, se procederá a la correspondiente verificación de los datos. Especialmente, se realizarán los pasos comentados en el apartado 3.5.1 (Registro de una nueva credencial).

Si el paso anterior se lleva a cabo de manera correcta, la plataforma guardará el valor del ID que identifica a la credencial junto con su clave pública, la cual será asociada al usuario que solicita su registro.

- **Autenticación**

  De igual manera, llegados a este punto se dará por hecho que un usuario dado ya ha registrado una credencial asimétrica, cuya clave privada estará en el autenticador y la pública en la base de datos del servidor.

  Cuando el usuario solicita iniciar sesión, se comprobará que su usuario y contraseña es válido, pasando así a realizar el segundo factor de autenticación.
  En primer lugar, el cliente nos requiere un desafío (/webauthn/getGetArgs), donde la librería utilizada formará un objeto JSON que incluye el challenge y la lista de las credenciales (allowCredentials) asociadas al usuario, para que este se pueda autenticar con alguna de ellas.
  Una vez que el usuario haya hecho uso de un token de seguridad y firmado el desafío, enviará al servidor el objeto correspondiente (Assertion). Allí, la plataforma tendrá que recoger la clave pública almacenada en base al ID de la credencial que ha firmado el desafío.
  Así pues, tan solo faltaría hacer uso de la librería para verificar el Assertion recibido, donde se seguirán los pasos comentados en el apartado 3.5.2 (Verificar un ‘assertion’ de autenticación) y se comprobará que la firma del desafío se ha realizado con la clave privada correspondiente a la que está almacenada, al igual que se ilustraba en la figura 11.

```php
$dataToVerify = '';
$dataToVerify .= $authenticatorData;
$dataToVerify .= $clientDataHash;

$publicKey = \openssl_pkey_get_public($credentialPublicKey);
if ($publicKey === false) {
    throw new WebAuthnException('public key invalid', WebAuthnException::INVALID_PUBLIC_KEY);
}

if (\openssl_verify($dataToVerify, $signature, $publicKey, OPENSSL_ALGO_SHA256) !== 1) {
    throw new WebAuthnException('invalid signature', WebAuthnException::INVALID_SIGNATURE);
}
```

Ilustración 20 - Verificación de la firma

Tal y como se puede apreciar en la ilustración anterior, se hace uso de openssl para recuperar la clave pública y posteriormente utilizarla para verificar la firma recibida (natural de la criptografía de clave pública). En caso de que la verificación no sea correcta, habrá una excepción y el proceso de autenticación no se llevará a cabo.

4.4 Resultado

Así pues, una vez comentadas las responsabilidades y procesos que ocurren en cada uno de los elementos del esquema diseñado, se procederá en este apartado a comentar el resultado final, un ejemplo de uso de WebAuthn en la plataforma desarrollada, que nos ayudará a aclarar conceptos, comprobar los beneficios que podemos obtener y el nivel de dificultad de implementación.

En primer lugar, tal y como comentábamos anteriormente, tanto la aplicación como la base de datos asociada han sido dockerizadas, permitiendo así su uso de una manera más sencilla, sin tener que preocuparse de instalaciones ni configuraciones. Por lo tanto, los ficheros necesarios se han alojado en un repositorio de Github [20], donde se detallan los pasos a realizar para ejecutar el servicio en cuestión.

Entrando en materia, en las próximas líneas se mostrará un ejemplo usando la plataforma, en la que primero nos registraremos como usuario para después registrar una clave asimétrica asociada. Veámoslo de manera gráfica:

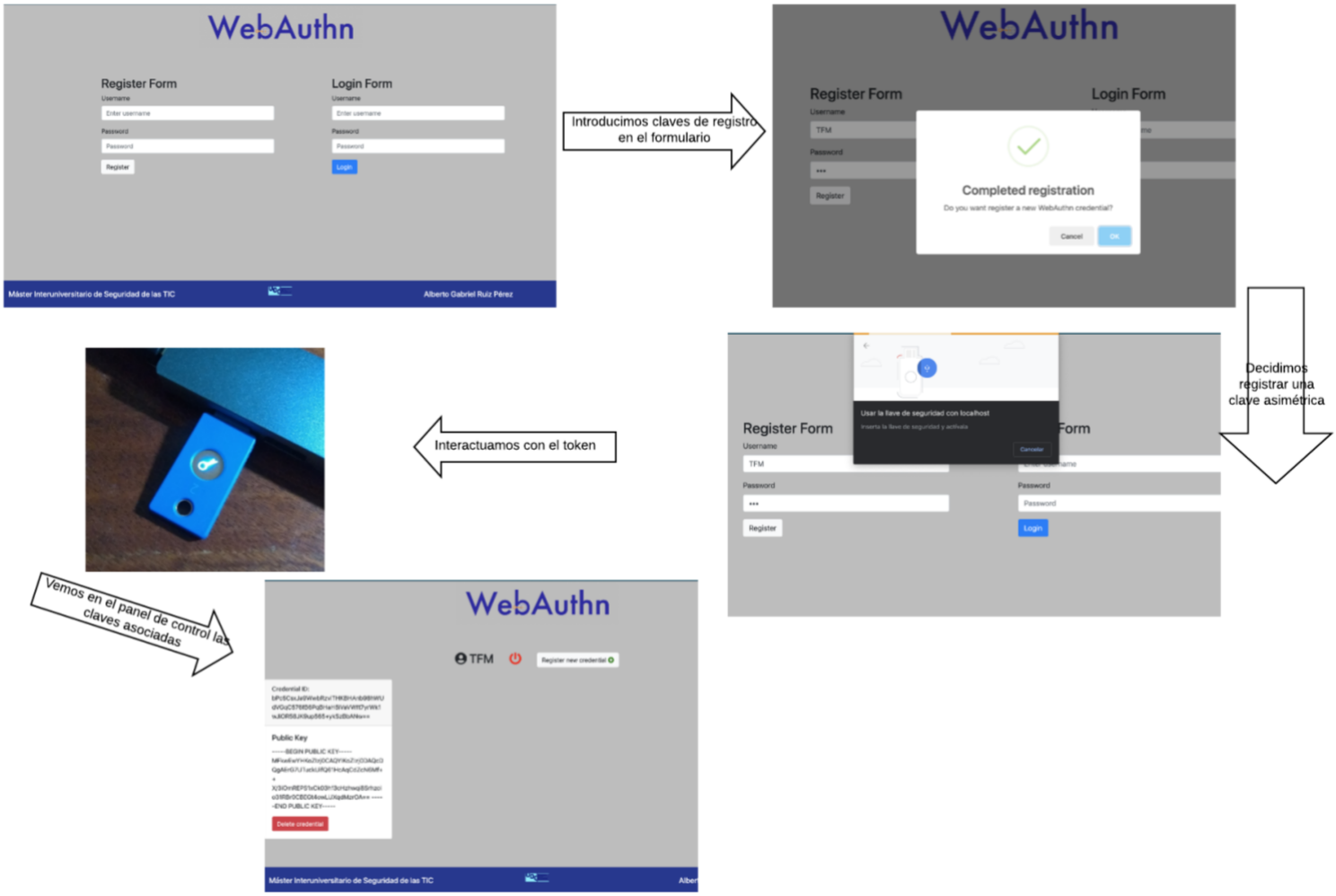


Ilustración 21 - Ejemplo de registro en la plataforma

Tal y como se ha podido reflejar en el diagrama anterior, vemos como en primer lugar se ha tenido que registrar a un usuario mediante la típica credencial Username+Password. Si esto se lleva a cabo (el servidor comprueba que no exista dicho usuario), se le preguntará al usuario si desea registrar una clave asimétrica, que en caso de que la respuesta sea afirmativa, se procederá a realizar lo detallado anteriormente en los puntos 4.2 y 4.3, donde en el lado del cliente los scripts harán uso de la API de WebAuthn y en el servidor de la librería empleada para tratar e interpretar los datos. En dicho proceso, la API solicitará la presencia de un token físico de seguridad, que interactuando con él daremos el consentimiento de la creación de una nueva credencial asimétrica.
Así, si esto se lleva a cabo, podremos loguearnos haciendo uso del token como 2FA, que nos dará acceso hacia nuestro panel de control donde se reflejarán las credenciales que han sido registradas.

De igual manera, en el panel de control de la sesión activa del usuario podremos realizar diversas acciones, como por ejemplo la que se muestra en la siguiente figura:

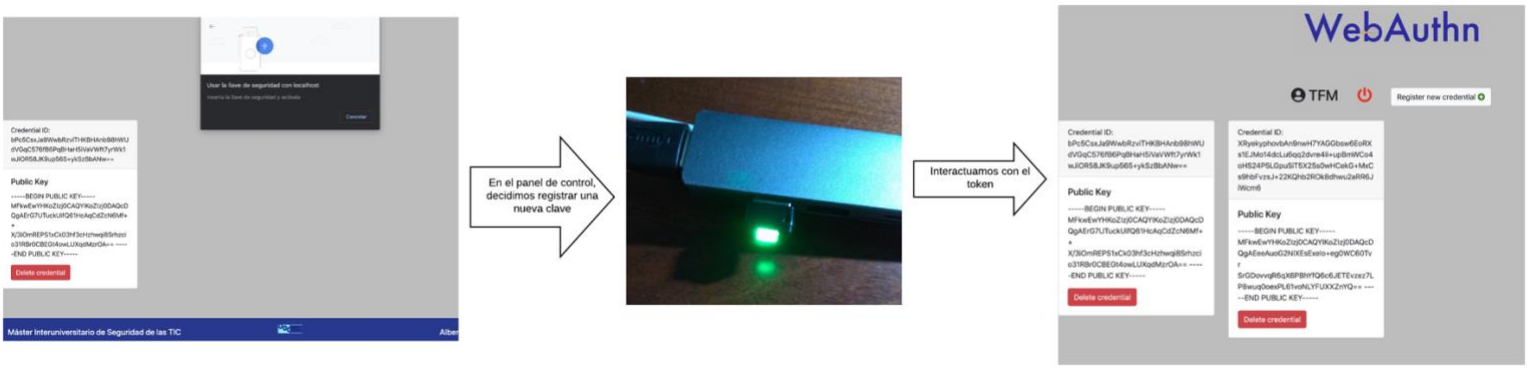


Ilustración 22 - Registro de una nueva clave en la plataforma

El diagrama anterior refleja que una vez el usuario tiene una sesión activa y una credencial asimétrica registrada en un token, decide que quiere registrar una nueva, pero esta vez utilizando un token físico diferente.
Como vemos, el proceso es idéntico al anterior caso, solamente que utilizaremos otro token, viendo así como se ha dado de alta la nueva credencial en el panel de control.
Así pues, en posteriores inicios de sesión, la plataforma solicitará la presencia de cualquiera de las dos credenciales, quedando bajo la decisión del usuario el utilizar uno u otro token.

Además como se puede ver, el panel de control nos da la posibilidad de eliminar e inhabilitar cada credencial, muy útil por ejemplo en casos de pérdida del token.

En conclusión, hemos podido ver como se ha incluido el estándar WebAuthn como método de segundo factor de autenticación, aumentando así la seguridad de las cuentas de usuario y combatiendo contra las amenazas que recaían sobre los anteriores métodos.
Cabe destacar, que a pesar de que en este caso se ha empleado como 2FA, tal y como hemos destacado en el capítulo 3.6 (Posibles escenarios), WebAuthn se podría emplear como único factor de autenticación, donde no sea necesaria la existencia de contraseñas ni se requiera el envío de información secreta a través de la red (gran beneficio de WebAuthn y de la criptografía asimétrica).

5. Conclusiones

En un principio, pudimos ver el estado actual de los métodos de autenticación existentes en los servicios más utilizados, especialmente el nivel de seguridad que estos tenían. De igual modo, constatamos que a pesar de la evolución tecnológica de los servicios, muchos de ellos seguían siendo amenazados por los tradicionales ataques cibernéticos, donde las credenciales son el principal objetivo de los mismos.

Por lo tanto, dadas las circunstancias actuales en el panorama de las aplicaciones y seguridad web, realizamos un barrido por las posibles alternativas, donde aquellos métodos que hacen uso de segundos factores de autenticación (SMS, email, TOTP, etc) están cobrando cada día más importancia, pero aún así siguen estando influenciadas por los ataques más comunes.

En este momento entró en juego el nuevo estándar WebAuthn, un nuevo método de autenticación web que hace uso de criptografía asimétrica y tokens de seguridad.
Durante el desarrollo del trabajo y del documento, hicimos una descripción en profundidad de su modo de funcionamiento, de los recursos necesarios y de los posibles escenarios donde implementarlo.
Además, realizamos un ejemplo práctico, donde incluíamos este método de autenticación en una supuesta aplicación web, de tal forma que pudimos comprobar fielmente tanto la facilidad de implementación como de uso.

Así pues, tras ver el alcance y las ventajas y desventajas de este nuevo protocolo, pudimos concluir que estamos ante una revolución en cuanto a métodos de autenticación en la web nos referimos, y es que con la inclusión de WebAuthn en los navegadores, las contraseñas pasan a ser innecesarias, además de que en los procesos de registro y autenticación ninguna información confidencial (claves) es enviada a través de la red, eliminando así los tradicionales ataques sobre credenciales (man in the middle, fuerza bruta, fuga de información, etc).

Como líneas de trabajo futuras respecto a WebAuthn, para aumentar la seguridad de los usuarios en plataformas web, será altamente conveniente la estandarización y compatibilidad de este protocolo en la gran mayoría de los navegadores web, incitando así a los desarrolladores para su implantación en las plataformas existentes.
De igual manera, con el paso del tiempo será beneficioso la existencia de bibliotecas de WebAuthn en múltiples lenguajes de programación para el lado del servidor (para realizar el flujo de comunicaciones y verificaciones pertinentes), posibilitando aún más su inclusión en cualquier tipo de aplicación web.

6. Glosario

En el apartado 3.2 pudimos ver como se listaban una serie de términos relevantes al estándar de WebAuthn. Del mismo modo, en este apartado enumeraremos unas definiciones correspondientes a expresiones utilizadas durante este documento, es decir, no se limitarán únicamente al protocolo sobre el que trabajamos.

- **Criptografía asimétrica:** También llamada criptografía de clave pública, es un método criptográfico que hace uso de dos claves complementarias (pública y privada) para enviar información a otra persona (que posee otro par de claves). Lo que está cifrado con una clave, se tiene que descifrar con la otra y así de manera viceversa.
  Este método es utilizado tanto para enviar información confidencial (cifrando con clave pública y descifrando con la privada) como para firmarla y asegurar su integridad (se cifra/firma con la privada y se comprueba con la pública).

- **Autenticación:** Proceso por el cual se confirma que algo o alguien es quien dice ser (identidad).

- **Estándar:** Es un documento o forma de utilización definida por consenso y aprobado por una entidad reconocida, ofreciendo una serie de reglas y guías para utilizar de forma repetida.

- **Man in the middle:** Es un ataque por el cual una tercera persona (atacante) tiene la capacidad de leer, escribir y modificar información perteneciente a una comunicación entre dos víctimas.

- **Servidor web:** Programa informático del lado del servidor, que realiza conexiones con los clientes, aceptando peticiones y procesándolas para posteriormente enviar una respuesta HTTP.

- **Credencial:** Es una orden o documento que testifica o autoriza la identidad de un individuo sobre una determinada entidad. En este caso nos referiremos a ‘credencial’ a las respectivas claves que utiliza el usuario para autenticarse ante un servicio.

- **Autenticación en dos factores (A2F):** Es un método que pretende confirmar la identidad de un usuario haciendo uso de dos componentes diferentes, como puede ser ‘algo que saben’, ‘algo que tienen’ o ‘algo que son’.

- **OpenSSL:** Proyecto de software libre que consiste en un potente paquete de librerías y herramientas relacionadas con métodos criptográficos. Esta herramienta nos puede ayudar a implementar SSL o TLS en un supuesto servidor. En nuestro caso, este paquete es utilizado

por la librería de WebAuthn en el servidor para verificar las firmas en los procesos de registro y autenticación.

- **Docker:** Herramienta que nos permite automatizar y virtualizar el despliegue de aplicaciones en contenedores. Para ello hace uso de características de aislamiento del kernel de Linux (cgroups y namespaces) para permitir la ejecución independiente dentro de una misma instancia.

- **Javascript:** Es un lenguaje de programación interpretado, utilizado comúnmente en el lado de cliente (implementado en los navegadores), permitiendo mejorar la apariencia e interactividad de las páginas web con los usuarios.

- **Token físico de seguridad:** Es un aparato electrónico que tiene el usuario para facilitar el proceso de autenticación sobre un servicio. En nuestro caso, estos dispositivos son utilizados para generar y almacenar claves asimétricas.

- **Symfony:** Framework PHP para el desarrollo de aplicaciones web basándonos en el patrón MVC. Además, nos provee varias herramientas y clases para agilizar el desarrollo, automatizando muchas de las tareas comunes.

- **MySQL:** Es un sistema gestor de bases de datos relacionales de código abierto. Actualmente es utilizado por la gran mayoría de aplicaciones y servicios web, que en nuestro caso se utilizará para almacenar la información relativa a los usuarios de la plataforma.

7. Bibliografía

[1] “Cambia tus contraseñas: filtrado el mayor robo de usuarios y ‘passwords’ de la historia”: https://hipertextual.com/2019/01/collecion-robo-contrasenas (Consultado: 25/03/2019).

[2] “Crece el número de robo de contraseñas en las bancas electrónicas: qué tenés que saber para protegerte”: https://www.clarin.com/tecnologia/crece-numero-robo-contrasenas-bancas-electronicas-tenes-saber-protegerte_0_4lHjTNvAp.html (Consultado: 25/03/2019).

[3] “WebAuthn obtiene la luz verde del W3C: las contraseñas de texto se aproximan a su fin”: https://hipertextual.com/2019/03/webauthn-es-aprobado-que-dejes-usar-contrasenas-mejore-tu-seguridad (Consultado: 25/03/2019).

[4] GanttProject: https://www.ganttproject.biz

[5] Yubiko Security Key: https://www.yubico.com/product/security-key-by-yubico/

[6] HYPERSECU HyperFIDO Mini: https://hypersecu.com/products/hyperfido

[7] Client to Authenticator Protocol (CTAP): (https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html) (Consultado: 25/03/2019).

[8] Web Authentication: An API for accessing Public Key Credentials: (https://www.w3.org/TR/webauthn/) (Consultado: 25/03/2019).

[9] WebAuthn, Wikipedia: https://en.wikipedia.org/wiki/WebAuthn

[10] Introduction to WebAuthn API: https://medium.com/@herrjemand/introduction-to-webauthn-api-5fd1fb46c285

[11] CBOR Object Signing and Encryption (COSE): https://www.iana.org/assignments/cose/cose.xhtml#algorithms

[12] WebAuthn Guide: https://webauthn.guide/

[13] WebAuthn/FIDO2: Verifying responses: https://medium.com/@herrjemand/verifying-fido2-responses-4691288c8770

[14] WORKSHOP: Authenticating your web like a boss: https://slides.com/fidoalliance/jan-2018-fido-seminar-webauthn-tutorial

[15] FIDO Alliance: WebAuthn Overview: https://slides.com/fidoalliance/webauthn-overview

[16] WebAuthn Awesome: https://github.com/herrjemand/awesome-webauthn

[17] Compatibilidad navegadores con WebAuthn: https://caniuse.com/#search=webauthn

[18] lbuchs/WebAuthn: https://github.com/lbuchs/WebAuthn

[19] Repositorio TFM: https://github.com/alber7rp/TFM-WebAuthn

[20] Repositorio TFM, plataforma web: https://github.com/alber7rp/TFM-WebAuthn/tree/master/code