Universidad Nacional de Río Negro
Sede Atlántica

Licenciatura en Sistemas

Trabajo Final de Carrera

Firma Digital con Token para Aplicaciones Web

Alumno: Villanueva Javier O., jovillanueva@gmail.com

Director: Mg. Molinari Enrique P., emolinari@unrn.edu.ar

Viedma, Río Negro. Año 2021.

Resumen. En la actualidad, existen muchas alternativas para firmar digitalmente documentos con un dispositivo criptográfico. Sin embargo, cuando integramos un software firmador en una aplicación web, el resultado puede ser una experiencia poco amigable para el usuario. Algunas de las pocas alternativas que existían, como los Applets de Java, fueron discontinuadas. En el presente trabajo final de carrera se analizan diferentes ventajas y desventajas de los enfoques existentes, los problemas necesarios a resolver y se desarrolla una solución de Firma Digital para Aplicaciones Web expresada en modelos de arquitectura y especificaciones junto con la implementación de una aplicación firmadora. El resultado final permite firmar digitalmente documentos con Tokens USB de manera simple e integrada en aplicaciones web.

Palabras Clave: Firma Digital, Token USB, Aplicación Web, PKCS#11

Índice

1 Introducción

El presente documento es el Informe de Trabajo de Producción correspondiente al Trabajo Final de la Carrera Licenciatura en Sistemas de la Universidad Nacional de Río Negro. El informe aborda el desarrollo de un firmador digital de documentos integrable en aplicaciones Web junto con el diseño y especificación de la arquitectura de software necesaria para su correcto funcionamiento.

El documento abarca los siguientes apartados:

- Problema a Resolver: en él se encuentra el enunciado del problema y los objetivos que debe cumplir la solución presentada.
- Marco Teórico (estado de la cuestión): explica los conceptos y la terminología que será utilizada en el informe. También se describe por qué no es posible acceder al dispositivo criptográfico desde un navegador web. Por último, se evalúan distintas alternativas existentes para alcanzar la solución.
- Solución: se detalla el diseño y especificación de la arquitectura de software necesaria para resolver el problema por medio de diagramas. Del mismo modo, describe el funcionamiento de cada uno de los componentes de la plataforma de firma digital para aplicaciones web.
- Producto: indica cómo descargar la aplicación. Además, explica cómo debe integrarse la plataforma en un sistema existente para que pueda firmar digitalmente con tokens usb.
- Caso de éxito: detalla el caso de éxito del Poder Judicial de Río Negro, en donde actualmente funciona la plataforma de firma digital presentada como solución.
- Conclusiones y líneas futuras: expone los tópicos principales del informe en cuestión. Asimismo, menciona posibles mejoras que pueden tenerse en cuenta en futuras versiones.
- Referencias: lista la bibliografía utilizada para el desarrollo de la solución y la confección del informe.

2 Problema a resolver y objetivos

2.1 Definición del Problema

Desde el año 2001 cuando el Congreso de la Nación sancionó la ley de firma digital, la difusión de esta herramienta ha sido lenta y paulatina debido a cuestiones en el diseño de la infraestructura de firma digital y en los procesos de licenciamiento que fueron necesario mejorar. Actualmente puede usarse tanto en los organismos del estado como para los ciudadanos y las empresas. Por ejemplo, en el sector público para gestiones como constancias y trámites a distancia ante organismos del Poder Judicial, la AFIP[1], la ANSES[2], el PAMI[3] o los Registros de Propiedad del Automotor.

La firma digital garantiza los tres principios de seguridad, autenticidad, integridad no repudio, y adicionalmente entre sus ventajas, permite llevar a cabo una gestión despapelizada, ya que garantiza la validez jurídica y probatoria de los documentos firmados digitalmente, según la ley nacional 25.506. No obstante, hay cierta reticencia hacia la firma digital como a la tecnología en general.

Asimismo, su integración en los sistemas, aún sigue siendo una tarea poco sencilla. Particularmente en las aplicaciones web el firmado digital a través de un dispositivo criptográfico puede resultar en una experiencia ardua y no muy amigable para el usuario, debido a la escasa integración entre el dispositivo y el sistema en cuestión. Estos motivos suelen complicar el uso de la firma digital y en consecuencia pueden apegarnos nuevamente a la firma manuscrita sobre papel con sus falencias.

El propósito este trabajo de producción es resolver el problema mencionado en el párrafo anterior, y así permitir que una persona, desde su navegador web, pueda firmar digitalmente con un dispositivo criptográfico (Token USB[4]) múltiples documentos PDF[5] desde una aplicación web de manera simple y transparente, sin necesidad de descargar los documentos en su computadora en forma manual y luego volver a subirlos a la aplicación firmados.

[1] AFIP – Administración Federal de Ingresos Públicos.
[2] ANSES – Administración de la Seguridad Social.
[3] PAMI – Instituto Nacional de Servicios Sociales para Jubilados y Pensionados.
[4] USB –Siglas en inglés de Universal Serial Bus.
[5] PDF – Siglas en inglés de Portable Document Format. Es un formato de documento portátil para presentar e intercambiar documentos de manera fiable, independientemente del software, el hardware o el sistema operativo, (Adobe Acrobat).

2.2 Objetivos de la solución

Primeramente, para poder satisfacer al propósito mencionado en el punto anterior, debemos definir bien cuales son los objetivos que debe alcanzar la solución:

- Simple de utilizar: Es decir, que no requiera más que una selección múltiple de documentos a firmar y el ingreso del PIN[6] de seguridad del Token.
- Integración transparente en aplicación web: Se debe poder integrar de manera sencilla y práctica los sistemas web, facilitando su reusabilidad en distintos aplicativos.
- Permitir firmar muchos documentos en poco tiempo: Esto es, firmar muchos documentos ingresando el PIN del Token sólo una vez.
- Permitir que los documentos sean firmados por múltiples personas: Por lo tanto, la firma de una persona no debe sobrescribir las anteriores.
- Debe dejar un estampado, de manera visible en el documento PDF, el nombre y apellido de las personas que lo firmaron.
- Firmar sin necesidad de descargar manualmente los documentos en la computadora del cliente.

3 Marco teórico

3.1 Conceptos generales

A continuación, se detallan los conceptos y la terminología que serán mencionados en los próximos capítulos.

3.1.1 Criptografía

De acuerdo con el Diccionario de la Real Academia, la palabra criptografía proviene de la unión de los términos griegos “kryptós” y “gráphein” que significan “oculto” y “escritura” respectivamente, y la define como: “El arte de escribir con clave secreta o de un modo enigmático”.

Lucena (2021) sostiene que la Criptografía hace años dejó de ser un arte para convertirse un conjunto de técnicas relacionadas con la protección de la información

[6] PIN – Número de Identificación Personal, de las siglas en inglés Personal Identification Number.

frente a observadores no autorizados. Adicionalmente, destaca la diferencia entre Criptografía y Criptoanálisis, en el sentido de que éste último término se refiere a las técnicas que se usan para romper los códigos criptográficos.

En general, la criptografía se lleva a cabo por medio de dos procesos que utilizan algoritmos matemáticos y claves. El proceso de cifrado transforma el mensaje original en un texto cifrado, luego el proceso inverso, llamado descifrado, vuelve a transformar al mensaje cifrado en el mensaje original.

Según Wadhwa et al. (2013) las metas de la criptografía son cuatro:

- **Confidencialidad**: El mensaje debería poder ser leído sólo por los receptores de destino.
- **Integridad**: El mensaje que le llega al receptor debería ser exactamente igual al original enviado.
- **Autentificación**: Prueba que el emisor sea el original y no otro que se haga pasar por él.
- **Disponibilidad**: Las tres metas anteriores son requeridas para lograr una comunicación digital segura, pero el proceso de lograrlas no debería reducir la performance de las aplicaciones. De esta manera el proceso debería llevar una sobrecarga lo más baja posible (en términos de velocidad y memoria).

Por otro lado, existen dos categorías principales de la criptografía en función del tipo de claves de seguridad utilizadas para cifrar/descifrar los datos, estas son: cifrado simétrico y asimétrico.

3.1.2 Cifrado Simétrico

Utiliza una sola clave, por eso se lo conoce también como de clave única. Previamente el emisor y el receptor deben ponerse de acuerdo sobre una clave secreta (compartida). Dado un mensaje y una clave, el proceso de cifrado produce datos ininteligibles. El descifrado es la operación inversa, utilizando la misma clave se obtiene como resultado al mensaje original.

Figura 3.1. Procesos para criptografía simétrica. Fuente: Medina y Miranda (2015, p.3)

Debido a que toda la seguridad se centra en una clave, ésta tiene que ser difícil de adivinar. Algunos algoritmos conocidos que se utilizan en este tipo de cifrado son: DES, AES, 3DES, entre otros.

3.1.3 Cifrado Asimétrico

El cifrado asimétrico, también conocido como criptografía de clave pública (Public Key Cryptography, PKC), maneja dos tipos de claves. La clave pública, conocida por todos, que se usa para cifrar el mensaje y la clave privada, conocida sólo por el receptor, se utiliza para descifrar el mensaje. Ambas claves están relacionadas matemáticamente y, lo que se cifra con la clave pública, sólo puede descifrarse con su correspondiente clave privada.

Generalmente utiliza longitudes de claves mayores a la simétrica. Por ejemplo, en los algoritmos simétricos, una clave se considera segura a partir de los 128 bits de longitud, mientras que para los asimétricos se recomiendan seguras claves de al menos 2048 bits. Los algoritmos utilizados en este tipo de cifrado se basan en plantear al atacante problemas matemáticos difíciles de resolver, el más popular por su sencillez es RSA (abreviación de sus creadores Rivest, Shamir y Adleman), de acuerdo a lo expuesto por Lucena (2021).

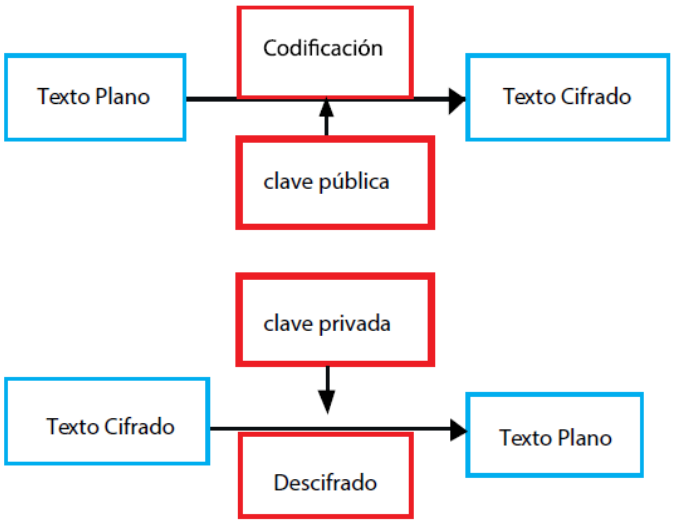

Figura 3.2. Procesos para criptografía asimétrica. Fuente: Medina y Miranda (2015, p.3)

3.1.4 Funciones de resumen

Una función de resumen, también denominada función hash, permite que, a partir de un mensaje de cualquier longitud, se pueda generar una secuencia de bits de longitud fija conocida como huella dactilar, resumen o digesto. El resumen obtenido es mucho más pequeño que el mensaje original, y es muy difícil encontrar otro mensaje diferente que dé lugar al mismo resumen, con lo cual, es muy difícil de falsificar. Principalmente, estas funciones se utilizan para detectar fallas de integridad de la información. Particularmente, dentro de las funciones hash, existe un tipo de funciones de resumen denominado MDC (del inglés Modification Detection Codes), que significa Códigos de Detección de Manipulaciones. Este tipo de funciones no necesitan de una clave para efectuar el cálculo y se utilizan para garantizar que un mensaje no haya sufrido modificaciones. Entre las funciones hash del tipo MDC más populares podemos encontrar a los algoritmos MD5 (del inglés Message-Digest Algorithm) y SHA-1/2/3 (del inglés Secure Hash Algorithm). (Lucena, 2012).

3.1.5 Firma Digital

3.1.5.1 Definición

Una firma digital es una secuencia de bits que se añade a una pieza de información cualquiera, y que permite garantizar su autenticidad de forma independiente del proceso de transmisión, tantas veces como se desee, (Lucena, 2021, p.

302). Por lo tanto, permite añadir a documentos digitales y mensajes de correo electrónico una huella o marca única.

La firma digital permite al receptor del mensaje o documento:

- Identificar al firmante de forma fehaciente (**Autentificación**).
- Asegurar que el documento no pudo ser modificado luego de la firma sin dejar evidencia de la alteración (**Integridad**).
- Tener garantías de que la firma se realizó bajo el control absoluto del firmante (**Exclusividad**).
- Demostrar el origen de la firma y la integridad del mensaje ante terceros, de modo que el firmante no pueda negar o repudiar su existencia o autoría (**No Repudio**).

3.1.5.2 Funcionamiento

Las firmas digitales se construyen en base a la encriptación asimétrica, utilizando una clave privada y otra pública. A diferencia de la criptografía de clave pública mencionada anteriormente, en la firma digital, ya no se busca encriptar el mensaje, sino darle una marca de autentificación. Para ello, la clave privada se utiliza en forma indirecta, es decir, no se aplica sobre el documento, sino sobre el resumen (digesto) obtenido del mismo luego de aplicarle una función hash.

Habitualmente la firma digital involucra dos procesos: la generación de la firma y su validación.

- **Generación de la firma**: el emisor A cifra el mensaje con una función hash de tipo MDC (ver 3.1.4), luego el digesto obtenido se encripta con la clave privada de A, obteniendo de este modo la firma del documento.
- **Validación de la firma**: el receptor B desencripta la firma con la clave pública de A, recuperando así al digesto. Adicionalmente, se vuelve a aplicar la misma función hash MDC al mensaje recibido, cuyo resumen resultante, se compara con el digesto anteriormente desencriptado. Si ambos resúmenes son idénticos, entonces el mensaje queda autenticado.

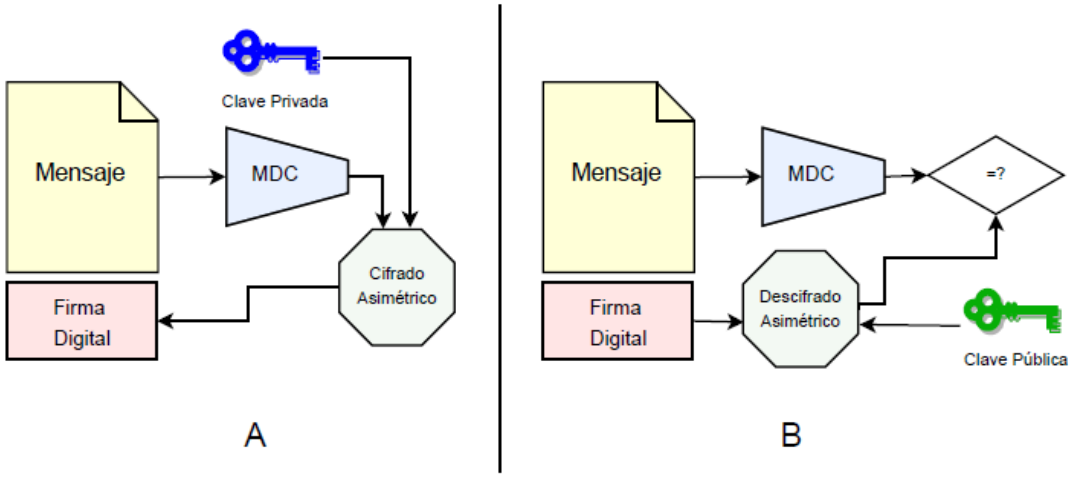

Figura 3.3. Esquema firma digital basada en funciones hash y cifrado asimétrico. La persona A firma el mensaje y la persona B lo verifica. Fuente: Lucena (2021, p.303)

3.1.5.3 *Firma Digital en Documentos PDF*

La norma ISO 32000 que define al estándar de archivos PDF tiene soporte para firmas digitales. En este tipo de archivo, la firma junto a más información relacionada a ella, quedan contenidas en una estructura llamada diccionario de firmas, dentro del archivo PDF. Cuando se firma un documento PDF, se computa el resumen/digesto sobre el rango de bytes (ByteRange) del archivo. Dicho rango está incluido en el diccionario de firmas, pero excluye a la firma en sí misma. Esto asegura que el digesto cubra todos los bytes del PDF, excepto a la firma del PDF. (ETSI TS 102 778-1, 2009). En la siguiente figura se observa como la firma del PDF es colocada en la entrada "/Contents" del diccionario de firmas.

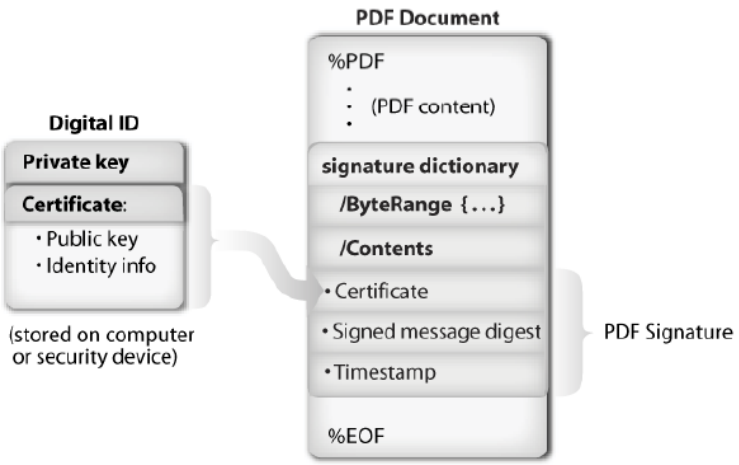

Figura 3.4. Estructura documento PDF firmado digitalmente. Fuente: ETSI TS, (2009, p.9)

Por otro lado, cada firma en el PDF puede contener un solo certificado. De este modo, si el documento necesita ser firmado más de una vez, el estándar lo resuelve de la manera mostrada en la siguiente figura. Para ello, se basa en que el documento puede tener tantos diccionarios de firmas como se desee y cada diccionario queda asociado con su propio ByteRage. Dicho en otras palabras, el segundo firmante, firma al documento junto con la firma de PDF previa, y así sucesivamente. Luego, la norma ISO 32000-1 estable que cada firma se verifica individualmente, pero el resultado final de la validación incluye a la de todas las firmas del documento.

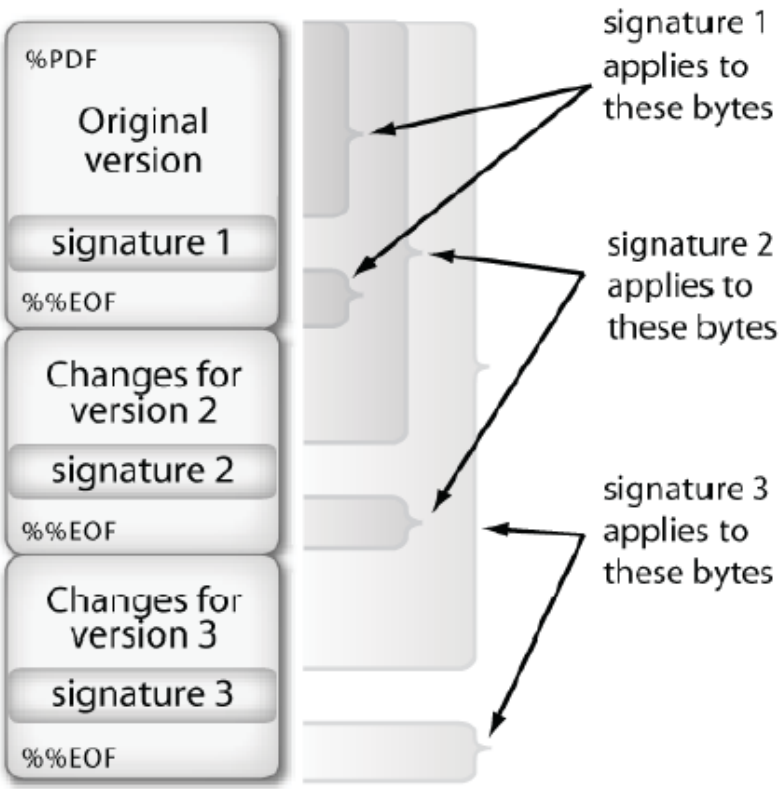

Figura 3.5. Estructura documento PDF con tres firmas digitales. Fuente: ETSI TS, (2009, p.9)

3.1.6 Certificados Digitales

Un certificado es una estructura de datos compuesta por una clave publica y un identificador que corresponde a la persona que posee la clave privada. Cada certificado de clave pública es emitido a un individuo y está compuesto por la firma digital de Autoridad Certificante que lo emitió, (Albarqi et al., 2015).

Principalmente los certificados se utilizan para demostrar que una clave pública pertenece a una persona en concreto. El estándar de certificado más común y extendido

en la actualidad es el X.509. Dicho formato se encarga de definir la sintaxis de los mismos y no está atado a ningún algoritmo en particular.

3.1.7 Infraestructura de Clave Pública

3.1.7.1 Definición

Para que el procedimiento de firma sea confiable, necesitamos saber con seguridad que la clave pública realmente pertenezca al firmante. La Infraestructura de clave Pública, (Public Key Infraestructure, PKI, en inglés), conforma uno de los elementos más importantes que sostienen al sistema de firma digital. Se encarga de regular la emisión y distribución de los certificados digitales que esencialmente mantienen datos de identidad de la persona, su clave pública y el nombre de la autoridad que emitió el certificado.

Por tanto, Albarqi et al. (2015) sostienen que las operaciones fundamentales de la Infraestructura de Clave Pública son la **certificación** con la que vincula un valor de clave pública a una entidad de una manera auténtica y la **validación** que hace referencia al proceso de verificar la validez de los certificados. Adicionalmente, mencionan que una Infraestructura de Clave Pública completa está compuesta por los siguientes componentes:

- **Autoridad Certificante (AC)**: es el emisor de los certificados y de las listas de certificados revocados. Los certificados tienen un tiempo de vida en el cual se consideran válidos (oscila entre uno y cinco años) y pueden ser revocados por varias razones como por ejemplo pérdida de la clave privada o porque se ha comprometido la clave y ya no es segura. Si algo de esto sucediera, la entidad emisora debe invalidar el certificado revocándolo. El mecanismo que utiliza para tal fin es la lista de certificados revocados (Certificate Revocation List, CRL, en inglés), que contiene la lista de los certificados que han sido revocados y está firmada digitalmente por la entidad que emitió a dichos certificados.
- **Autoridad de Registro (AR):** es una entidad intermediaria entre la persona y la Autoridad de Registro. Se usa para enviar todas las solicitudes de certificados a la AC, autenticar a las identidades de los usuarios y registrar toda la información del usuario antes de la certificación.

- **Repositorio de Certificados y Sistema de Distribución:** usados para proveer un mecanismo de almacenamiento seguro para la información de los certificados y las CRL.
- **Certificados Digitales:** los certificados definidos en el punto anterior.
- **Usuarios:** son las personas que pueden firmar digitalmente.

3.1.7.2 Regulación

Si el certificado es auténtico y confiamos en la autoridad emisora, podemos asegurar la identidad del firmante. En nuestro país, esta regulación se conoce como Infraestructura de Firma Digital de la República Argentina (IFDRA) y permite la emisión de certificados, para que luego, se puedan verificar las firmas en condiciones seguras, tanto desde el punto de vista técnico como legal.

La ley Nº 25.506 sanciona que los procedimientos de firma y verificación deben ser determinados por una Autoridad de Aplicación. Actualmente, dicho rol es desempeñado por la Subsecretaría de Innovación Administrativa, que depende de la Jefatura de Gabinete de Ministros. El mencionado organismo actúa como Ente Licenciante y es el encargado de otorgar, denegar o revocar las licencias de los certificadores licenciados, y adicionalmente, supervisa su accionar.

3.1.7.3 Jerarquía de Certificados

Parar poder constatar que una firma fue emitida realmente por una Autoridad de Certificación, también se emiten certificados digitales a dichas autoridades. Estos certificados, a su vez, están firmados por una entidad de mayor jerarquía. De esta manera se genera una "cadena de confianza". Luego, adquiriendo el certificado de la autoridad máxima se puede validar sucesivamente los certificados de menor jerarquía.

La Infraestructura de Clave Pública de la República Argentina define dos niveles de autoridad:

- **Autoridad certificante Raíz de la República Argentina (AC-RAIZ)**: es la de mayor jerarquía en la IFDRA y está representada por el Ente Licenciante mencionado en el punto anterior. Este certificado denominado Raíz es necesario para validar cualquier firma digital y está firmado por la propia autoridad.

Asimismo, se encarga de emitir los certificados digitales a las Autoridades Certificantes de segundo nivel.

- **Certificadores Licenciados**: Son conocidos como Autoridades Certificantes de segundo nivel y considerados Certificados Intermedios. Operan con las Autoridades Certificantes y las Autoridades de Registro. Son necesarios para poder validar las firmas de las personas que hayan obtenido la firma digital a través de ellos. Algunos de los certificadores licenciados con licencia vigente:
  - ONTI – Oficina Nacional de Tecnologías de Información
  - Ministerio de Modernización

3.1.8 Dispositivos Criptográficos

3.1.8.1 Características

Es imperioso que, al trabajar con certificados digitales, la clave privada del firmante quede almacenada en un lugar seguro. El objetivo es prevenir que una persona mal intencionada pueda firmar a nombre de otra robando la identidad digital del verdadero dueño del certificado.

Los dispositivos criptográficos, también conocidos como tokens, proporcionan un almacenamiento seguro para certificados digitales y claves privadas. De esta manera, permiten que la criptografía de clave pública y las firmas digitales se usen de forma segura sin riesgo de filtrar información de la clave privada. Adicionalmente agregan mayor nivel de seguridad al proveer autenticación multi-factor al incorporar “algo que el usuario conoce”, “algo que el usuario posee” y “algo que el usuario es”.

En general, los tokens poseen su propio sistema operativo y motor criptográfico, permitiendo generar claves y firmar digitalmente entre otras operaciones algorítmicas complejas. Son dispositivos robustos, con capacidad de autobloqueo ante un ataque de fuerza bruta, a diferencia de otros dispositivos de almacenamiento como pendrives, computadoras personales, notebooks, etc.

En el mercado pueden encontrarse diferentes tipos de Tokens: Tokens USB, Tokens OTP (One-Time-Password), SmartCards (Tarejetas Inteligetes), hasta inclusive BioTokens capaces que almacenar la huella digital. Cada tipo de dispositivo suele proveer soluciones a distintos problemas de seguridad. Como se mencionó

anteriormente, el problema a resolver en el presente trabajo, requiere el firmado a través de Tokens USB.

Para poder firmar con un Token USB, primeramente, es necesario tener instalado un software middleware específico provisto por la empresa proveedora del token. Luego, la computadora debe reconocer al dispositivo criptográfico conectado al puerto USB. Por último, se deberá ingresar el PIN/Password de acceso al token para que éste puede firmar digitalmente utilizando la clave privada del certificado almacenado de manera segura en el dispositivo.

Figura 3.6. Token USB. Fuente: MacroSeguridad.

3.1.8.2 El estándar PKCS #11

Los Estándares de Criptografía de Clave Pública (conocidos como Public-Key Cryptography Standars, PKCS en inglés) comprenden un grupo de estándares criptográficos que provén guías y APIs[7] para el uso de métodos criptográficos. Una de las principales metas de estos estándares es poder tener distintas implementaciones interoperables, es decir que puedan comunicarse entre sí y compartir información.

En especial, el estándar PKCS#11, define una API multiplataforma llamada "Cryptoki", de Cryptographic Token Interface, para la comunicación con dispositivos como los tokens que almacenan información criptográfica y ejecutan funciones criptográficas. El estándar Cryptoki permite gestionar el hardware criptográfico aislando a las aplicaciones de los detalles del dispositivo criptográfico, para poder lograr independencia tecnológica (cualquier tipo de dispositivo hardware) y uso compartido de recursos. (Yongge, 2012). Generalmente, la API es implementada por librerías dinámicas que luego son utilizadas y compartidas por las aplicaciones. A continuación,

[7] Una API (en inglés Application Programming Interface) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.

se muestra un modelo genérico, en donde se aprecian los distintos niveles que atraviesan las llamadas de la aplicación hasta llegar al token.

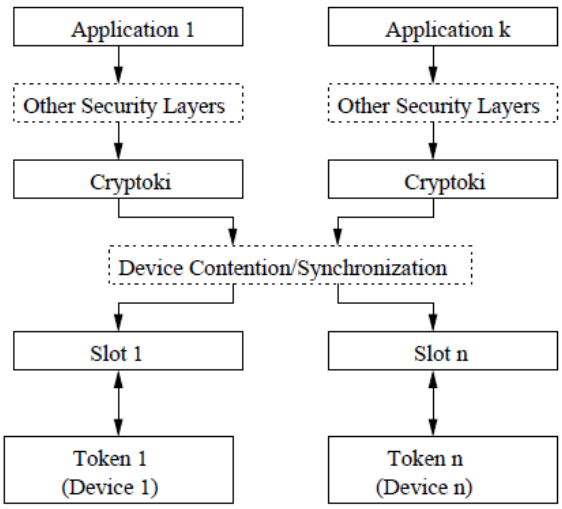

Figura 3.7. Modelo Genérico Cryptoki. Fuente: Yongge (2012, p.10)

3.2 Acceso al token desde el navegador

Retomando con la problemática planteada en este trabajo, es primordial resolver el problema de firmar con un token USB desde una aplicación web. Para ello, se analizará en esta sección la posibilidad de poder firmar accediendo al dispositivo criptográfico desde el mismo navegador web.

De acuerdo con lo expuesto por Ferri et al. (2010 pp. 1, 6) los navegadores web deben instanciarse dentro de un ambiente virtual controlado denominado sandbox, cuyo espacio de memoria y recursos son designados de manera restringida. De tal forma, no comprometemos al sistema anfitrión durante la navegación por sitios web riesgosos. Esto se logra abriendo los hipervínculos en sesiones sandbox que virtualizan algunos recursos para que más tarde, al cerrar la sesión, los cambios efectuados dentro de dicho ambiente no persistan en el sistema anfitrión, aunque, por cierto, el usuario puede llegar a configurar cierto grado de persistencia. Este mecanismo de seguridad es implementado actualmente por todos los browsers[8] modernos como Google Chrome, Mozilla Firefox, Microsoft Edge y Chromiun.

[8] Browser - Término en inglés que se utiliza para identificar a un navegador web o de internet. Conociste en un software que básicamente permite visitar páginas web.

Más específicamente, para el caso de Google Chrome, según Reis et al. (2009 pp. 1-3), se utilizan sandboxes para limitar el daño que pueda causar un atacante intentando explotar alguna vulnerabilidad en el motor de renderizado. Se implementa a través de varias capas de defensa. Primero, el contenido web se ejecuta dentro de una máquina virtual JavaScript que actúa de sandbox. Luego, utiliza una capa que actúa de barrera para exploits[9], haciendo aleatorio el espacio de direcciones. Finalmente, en la capa más baja, utiliza la implementación sandbox provista por el sistema operativo para limitar el accionar del proceso y evitar que este cause daños. La arquitectura de Google Chrome coloca al proceso que ejecuta el motor de renderizado en un sandbox con muy bajo nivel de privilegios, a diferencia de su kernel que corre en otro proceso con mayor nivel de privilegios. De esta manera, el sandbox provisto por el sistema operativo previene que el motor de renderizado interactúe directamente con el sistema. Este mecanismo de seguridad bloquea el acceso a los archivos, dispositivos y otros recursos del sistema operativo anfitrión.

Similarmente, tal como mencionan Bhavaraju et al. (2018 p. 3), el navegador Mozilla Firefox también implementa su modelo de seguridad basándose en sandboxes. Aunque su implementación varía de acuerdo con el sistema operativo, la idea es tener un proceso principal padre con acceso al sistema operativo que sólo ejecuta código de confianza. Además, está compuesto por muchos procesos hijos, llamados Web Content Processes, que corren en sandboxes con un nivel mínimo de privilegio, destinados a ejecutar el código que no es de confianza. Luego, el proceso padre actúa de mediador entre el proceso hijo y el sistema operativo. En efecto, los procesos de tipo Web Content quedan con acceso muy limitado al sistema de archivos, no pueden ejecutar nuevos procesos y no pueden acceder a APIs y llamadas a sistemas que comprometan la integridad del sistema.

En conclusión, de acuerdo a lo expuesto, no es viable firmar desde el navegador con una aplicación web accediendo directamente al token del usuario. El problema radica en que los mecanismos de seguridad de los navegadores modernos no lo permiten.

[9] Un exploit es un tipo de programa creado para sacar provecho de una vulnerabilidad de seguridad determinada de un software o hardware.

3.3 Alternativas existentes

En esta sección, se analizan distintas alternativas para resolver el problema. Las soluciones se dividen en dos tipos de arquitecturas: firmar del lado del cliente o del lado del servidor.

3.3.1 Firmar en el Cliente

En este tipo de arquitectura, la aplicación encargada del firmado corre en la computadora del cliente. La siguiente figura muestra dos esquemas, en el de la izquierda, la firma (Signed Message Digest, el digesto del mensaje firmado), se crea en el dispositivo criptográfico, por ejemplo, en un Token USB o Smart Card[10], sin embargo, en el esquema de la derecha la firma es calculada por software.

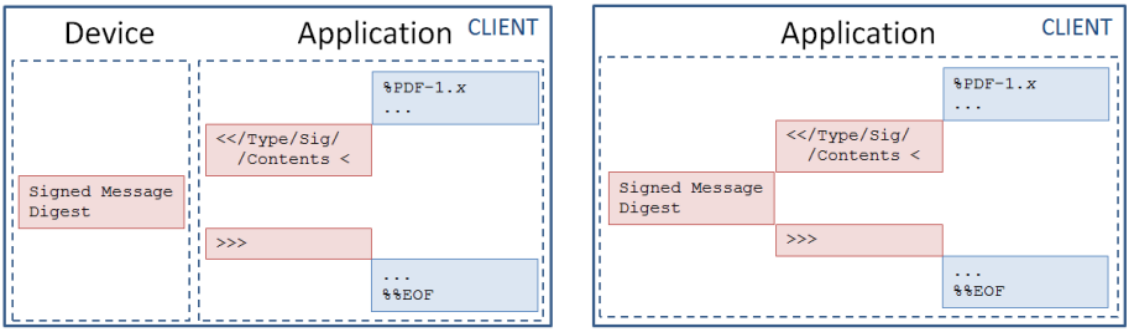

Figura 3.8. Soluciones del lado del Cliente. Fuente: Lowagie (2012)

Para nuestro caso en concreto, corresponde el esquema de la izquierda. A continuación, se analizan distintas alternativas que se encuadran en dicho esquema.

3.3.1.1 Aplicaciones de Escritorio

Las aplicaciones de escritorio no poseen las restricciones de seguridad que imponen los navegadores de internet. Sin embargo, tienen la desventaja de que no suelen integrarse del todo bien a las aplicaciones web. Generalmente el usuario tiene que descargar los documentos desde la aplicación web, firmarlos y luego volver a subirlos a ella. Dicha manera de operar es incompatible con los requerimientos expuestos anteriormente.

[10] Las Smart Cards (Tarjetas Inteligente) tienen muchas utilidades, aquí se hace mención a su uso como token criptográfico, es decir, como dispositivo que permite procesar o portar información personal, como generar y guardar las claves. Físicamente, una Smart Card es una tarjeta de material plástico.

Existen algunas herramientas como XolidoSign que en su versión gratuita permite firmar múltiples documentos a la vez. Adicionalmente, ofrece versiones pagas que proveen distintas funcionalidades que incluyen integración con aplicaciones web, sin embargo, la firma debe ser a través de certificados digitales y no permite el uso de dispositivos criptográficos como los tokens.

3.3.1.2 Applets de Java.

Un applet de Java es un componente escrito en Java que se ejecuta sobre un contenedor dentro del navegador web. Por ejemplo, SIU-Toba es un applet muy conocido que permite firmar digitalmente y pertenece al Sistema de Información Universitaria integrado por Universidades Nacionales Públicas. Entre sus ventajas podemos mencionar que es open-source, funciona con tokens y permite firmar múltiples documentos a la vez. Sin embargo, la desventaja de este enfoque reside en que, para que el applet se ejecute, es necesario que el navegador tenga instalado el Plugin de Java. Pero sucede que dicho plugin funciona sobre una arquitectura NPAPI (Netscape Plugin Application Programing Interface) que los browsers han dejado de dar soporte para unificar las funcionalidades entre las versiones desktop y mobile (Oracle, 2016). Sin ir más lejos, el soporte que provee Mozilla Firefox es muy limitado para las versiones de 32 bits y lo abandonó para las de 64 bits, por otro lado, Google Chrome finalizó el soporte de esta tecnología en el año 2015. Finalmente, Oracle discontinuó los Applets desde Java SE9 (Oracle, 2020).

3.3.1.3 Extensiones/Addons de navegadores.

Otra alternativa posible es desarrollar una extensión, también conocida como complemento o Add-On, para algún browser específico como Mozilla Firefox o Chrome. La extensión, una vez instalada en el navegador, hace de nexo entre la sesión de la aplicación web y el entorno del sistema operativo, logrando el acceso al dispositivo de firma digital. Si bien la firma se produce de manera casi directa, la desventaja está en que los usuarios del sistema quedan limitados a usar siempre un mismo navegador. O nos obliga a desarrollar el mismo Add-On para todos los proveedores de navegadores desde donde se acceda al sistema.

3.3.2 Firmar en el Servidor

Hasta aquí, los enfoques mencionados, realizan el firmado del lado del cliente. Sin embargo, también existe la posibilidad de realizar la firma en el servidor (en lugar de realizarla en el cliente con Token). Esto es posible únicamente si los certificados (con las claves privadas) de los firmantes son almacenados en el servidor de manera segura.

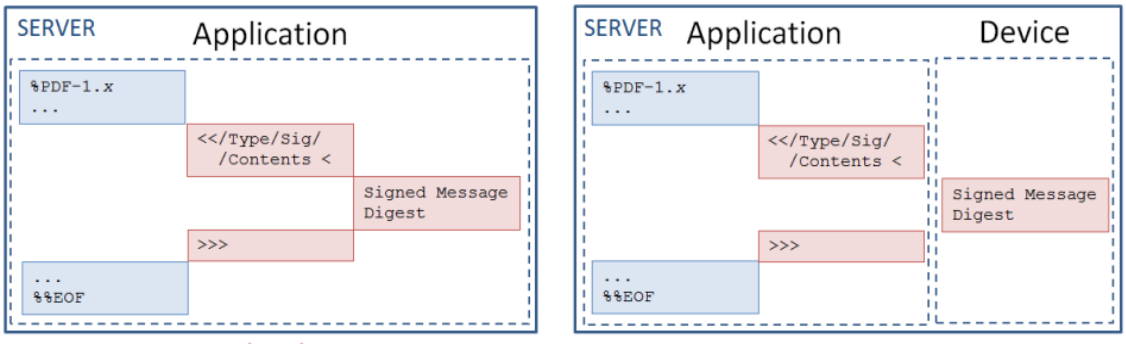

Figura 3.9. Soluciones del lado del Servidor. Fuente: Lowagie (2012)

Debido a que esta arquitectura se adapta mejor cuando los documentos necesitan ser firmados por compañías en vez de individuos con sus tokens personales, según Lowagie (2012), no es la adecuada para resolver el problema que se planteó.

A modo de ejemplo, esta posibilidad la tiene el estado nacional y para ello la Jefatura de Gabinete de Ministros implementó la Plataforma de Firma Digital Remota (PFDR). Dicha plataforma permite al usuario subir documentos en PDF y firmarlos digitalmente allí, ingresando un OTP (One Time Password) como mecanismo adicional de seguridad que el sitio envía al teléfono del firmante para asegurar la autenticidad, y luego ingresar el PIN. La PFDR, a través de un conjunto de APIs, permite una buena integración con aplicaciones web de terceros, pero actualmente con la limitación de firmar de a un documento a la vez. A su vez, como mencionamos, era necesario firmar muchos documentos PDF en poco tiempo, lo que no es factible de hacer con una solución remota.

4 Solución - Plataforma de Firma Digital para Aplicaciones Web

El desarrollo de la solución que se presentará a continuación, se fundamenta en la arquitectura expuesta por Villanueva y Molinari (2021). Dicha arquitectura establece

un firmador en donde la firma con el Token USB se efectúa del lado del cliente. Por lo tanto, es necesario desarrollar una aplicación de escritorio para que el usuario firme. Sin embargo, para tener una integración transparente con la aplicación web, y permitir que pueda adaptarse a distintos aplicativos, es necesario agregar a la implementación algunos componentes de software adicionales.

Para describir y comunicar la arquitectura la Plataforma de Firma Digital se utilizarán los modelos propuestos por Simon Brown denominados C4 Model. En concreto, utilizaremos el modelo de contenedores y el de componentes. Los contenedores (no confundir con Docker) del modelo C4 se refieren a aplicaciones (Web, móbiles, single-page, de escritorio, shell scripts, etc), repositorios de datos, esquemas y bases de datos de cualquier tipo, relacionales o no. Esencialmente es un elemento de software que ejecuta código o almacena datos. Con componente, el modelo C4 se refiere a un conjunto de funcionalidades encapsuladas y expuestas a través de una interfaz bien definida. Éstos deben mapear a una estructura sintáctica como un package, espacio de nombres o módulos. El diagrama de componentes es un nivel de detalle superior al de contenedores. Dentro de un contenedor, encontraremos uno o varios componentes.

4.1 Diagrama de contenedores

Inicialmente, se plantea un diagrama de contenedores para describir la plataforma de una forma más abstracta. El siguiente modelo está compuesto por cuatro contenedores: la Aplicación Web, el Firmador, el Módulo de Documentos Temporales y el Web Browser. A continuación, se describen dichos contenedores y sus relaciones:

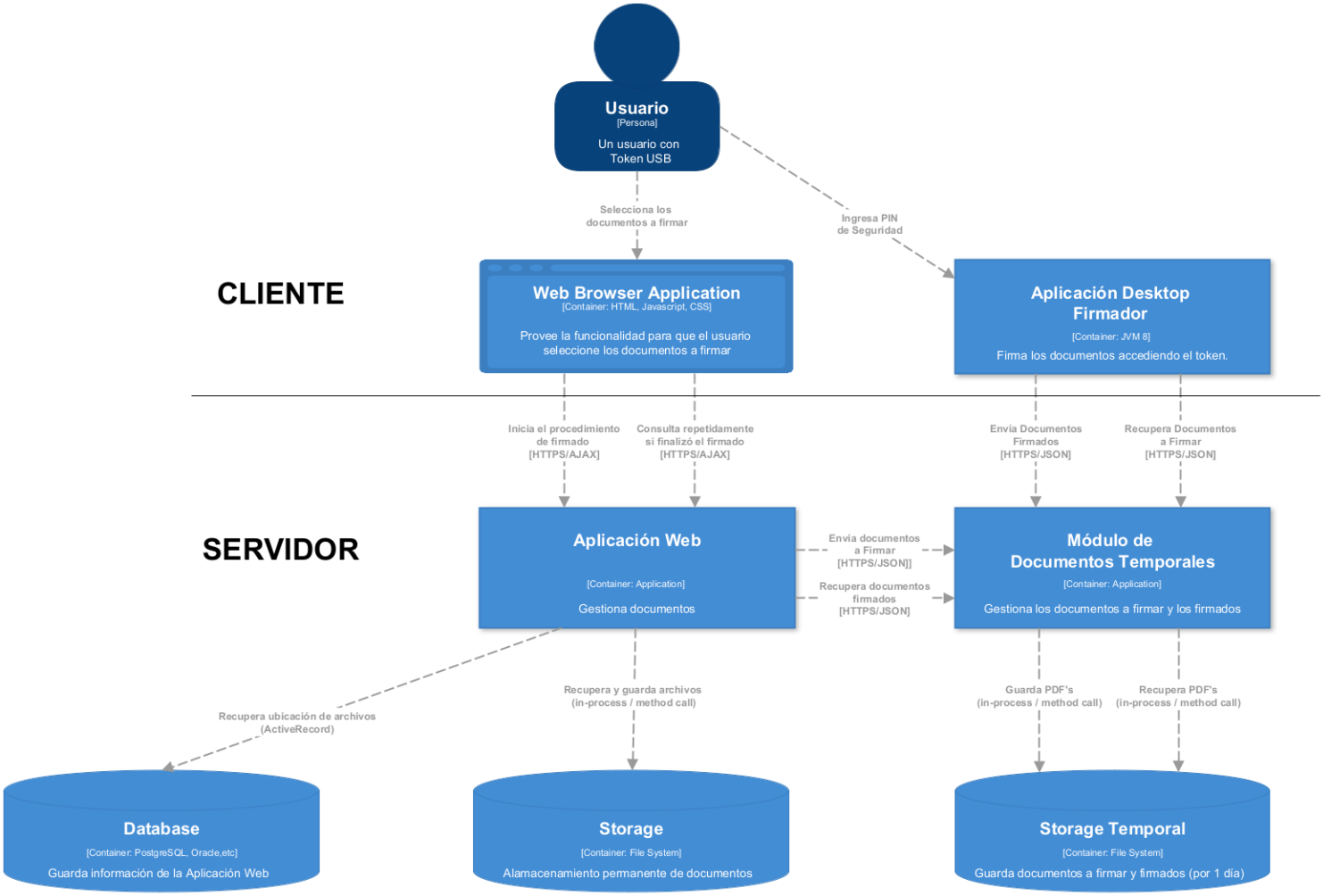

Figura 4.1. Diagrama de Contenedores - Plataforma de Firma Digital para Aplicaciones Web

El usuario, desde su navegador de internet, accede a la Aplicación Web que, en nuestro caso, representa a un aplicativo de gestión que requiere firmar digitalmente documentos PDF. Luego, desde el browser, la persona selecciona los documentos a firmar, y cuando presiona el botón de firmar, el navegador le ordena a la aplicación web que inicie el proceso de firmado indicándole que archivos firmar. El navegador, además, inicia la ejecución de la aplicación desktop firmadora Java como se verá en detalle más adelante.

La aplicación web se comunica con el Módulo de Documentos Temporales (MDT) para enviarle los documentos en formato PDF a firmar y luego para recuperarlos firmados. Adicionalmente avisará al navegador cuando el proceso de firmado haya finalizado.

El Módulo de Documentos Temporales es una aplicación mediadora. Se ejecuta en un contenedor separado del sistema anfitrión en un mismo server o diferente. Básicamente mantiene los documentos en PDF que se van a firmar y los que ya fueron firmados de manera temporal. El módulo expone un conjunto de APIs REST[11] utilizadas por la aplicación desktop firmadora y por la aplicación web.

El contenedor Aplicación Desktop representa a la aplicación firmadora Java. Se ejecuta en la PC del usuario y se encarga de descargar los documentos a firmar para luego enviárselos firmados al Módulo de Documentos. Adicionalmente, le solicita al usuario que ingrese el PIN de seguridad del su Token para luego iniciar el proceso de firmado.

El contenedor Aplicación Desktop se ejecuta en la computadora del usuario y se encarga de firmar digitalmente documentos con formato PDF. Para ello, primero le solicita el PIN del token al usuario, luego descarga los documentos del MDT, los firma, y finalmente envía los archivos firmados al MDT.

4.2 Componentes de la plataforma

Se presenta a continuación un diagrama de componentes para describir el diseño de la arquitectura anterior con mayor detalle. De este modo, el siguiente modelo muestra los componentes de cada uno de los contenedores que implementan la solución.

[11] REST - Es un acrónimo de Representational State Transfer o Transferencia de Estado Representacional, le agrega una capa muy delgada de complejidad y abstracción a HTTP. Mientras que HTTP es transferencia de archivos, REST se basa en la transferencia de recursos.

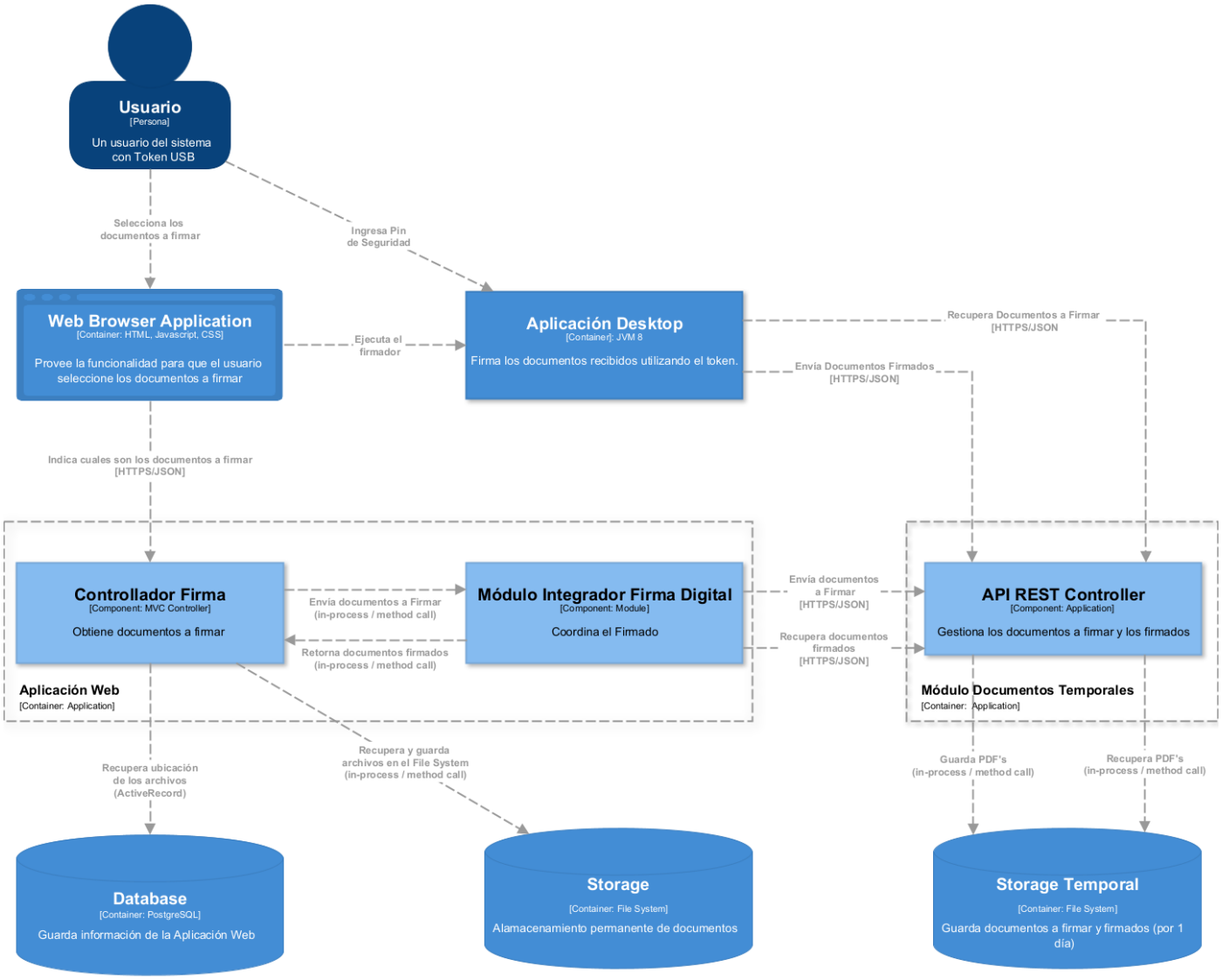

Figura 4.2. Diagrama de Componentes - Plataforma de Firma Digital para Aplicaciones Web

4.2.1 Aplicación Web

El sistema que implementa a la aplicación web recibe la acción de firmado que proviene del navegador, por medio de una llamada HTTP POST que se dispara cuando el empleado presiona el botón de firmar. Dicha llamada es manejada por un componente controlador específico encargado de iniciar el procedimiento de firmado. La aplicación web, desde el lado del servidor, debe coordinar y sincronizar algunas de las tareas que efectúan el módulo de documentos, el firmador de escritorio y el navegador. La creación del Módulo Integrador de Firma Digital (MIFD) para Aplicaciones Web representa una solución que permite extraer dicha funcionalidad e independizarla para

poder reutilizarla en otras aplicaciones que requieran firma digital. Este módulo representa un componente integrador que se encarga de indicar qué hacer en cada momento y efectuar el vínculo con nuestra aplicación web.

El módulo MIFD es iniciado por el controlador del sistema ya mencionado y se encargada de varias tareas. Implementa la comunicación entre la aplicación en el servidor y el navegador en el cliente. Primero, envía los archivos PDF para firmar al módulo de documentos. Luego, el navegador se queda esperando en un bucle en el que repetidamente le consulta al MDT si recibió los documentos PDF ya firmados. Por último, cuando detecta que se cumple dicha condición, recupera los archivos firmados desde el módulo de documentos temporales, se los entrega al controlador para que los persista y le avisa al browser que el proceso de firmado finalizó con éxito.

4.2.2 Módulo de Documentos Temporales

El MDT es un módulo independiente del resto de los componentes, que actúa de mediadora entre la aplicación web y de la aplicación de escritorio firmadora. Se encarga de manejar los documentos en PDF que se van a firmar y los documentos que ya fueron firmados. Expone un conjunto de APIs REST utilizadas por la aplicación web y a la aplicación firmadora, que permiten gestionar el manejo de los documentos que se necesitan firmar, los firmados y el estado del proceso de la firma. Es importante resaltar, que cuando se inicia la tarea de firmado, se le asigna a ésta un identificador aleatorio (uuid) que representa a todos los archivos a firmar. Dicho identificador será definido de aquí en adelante, como identificador de carpeta.

Este módulo puede disponer de su propio storage para persistir temporalmente los documentos. Es importante resaltar que el almacenamiento de los documentos es temporal, puesto que luego de que el proceso de firmado finaliza, los documentos procesados son almacenados por la aplicación web en su propio storage. Por tal motivo dejan de ser necesarios en este módulo y deben ser eliminados gradualmente por un proceso batch diariamente.

4.2.2.1 APIs utilizadas por el firmador

A continuación, se especifican las APIs del MDT que son llamadas por el firmador Java, todas deben estar implementadas para que funcione la aplicación desktop firmadora.

<table>
<tr><th colspan="2">POST api_url/recuperar-a-firmar</th></tr>
<tr><td>Descripción</td><td>Dado un id de carpeta recupera los archivos a firmar asociados en ésta.</td></tr>
<tr><td>Parámetros de entrada</td><td>Request Body:
Formato: Json
Requerido: Sí
Schema: {‘Carpeta’: {‘id’: String} }</td></tr>
<tr><td>Authorization</td><td>Bearer xxxxxxxxxx</td></tr>
<tr><td>Headers</td><td>Content-Type: application/json</td></tr>
<tr><td>Salida</td><td>Schema: [{‘id': String, ‘arch': String (Base64)}]
<code>{"success":true,"data":[{"id":"id1","arch":"base64_1"}, ...]</code>
<code>{"success":false,"data":{...error...}}</code></td></tr>
<tr><td colspan="2"><pre>curl -X POST -H 'Content-Type: application/json' -H 'Authorization: Bearer xxxxxxxxxx' -i 'http://module_api_url/recuperar-a-firmar' --data '{
 "Carpeta": {
    "id": "uuid_carpeta"
  }
}'</pre></td></tr>
</table>

Tabla 4.1. Definición de la API REST /recuperar-a-firmar

<table>
<tr><th colspan="2">POST api_url/guardar-firmados</th></tr>
<tr><td>Descripción</td><td>Guarda en el módulo de documentos todos los archivos firmados asociados a un id de carpeta.</td></tr>
<tr><td>Parámetros de entrada</td><td>Request Body:
Descripción: Recibe el id de carpeta y todos los archivos firmados en Base64.
Formato: Json
Requerido: Sí
Schema:
{‘Carpeta’: {‘id’: String, ‘pdfs’: [{‘id’: String, ‘arch’: String (Base64)}] }}</td></tr>
<tr><td>Authorization</td><td>Bearer xxxxxxxxxx</td></tr>
<tr><td>Headers</td><td>Content-Type: application/json</td></tr>
<tr><td>Salida</td><td>Descripción: la cantidad de archivos guardados con éxito.
Schema: {‘cantidad': Integer}
{"success":true,"data":{"cantidad":1}}
{"success":false,"data":{...error...}}</td></tr>
<tr><td colspan="2">curl -X POST -H 'Content-Type: application/json' -i 'http://module_api_url/guardar-firmados' --data '{
"Carpeta": {
"id": "uuid_carpeta",
"pdfs": [{"id":"id1","arch":"base64_1"},
{"id":"id2","arch":"base64_2"}]
}
}'</td></tr>
</table>

Tabla 4.2. Definición de la API REST /guardar-firmados

POST api_url/finalice-firma	
Descripción	Para un id de carpeta, registra en el módulo de documentos que el firmador finalizó con su tarea.
Parámetros de entrada	Request Body: • Descripción: Recibe un id de carpeta. • Formato: Json • Requerido: Sí • Schema: {'Carpeta': {'id': String} }
Authorization	Bearer xxxxxxxxxx
Headers	Content-Type: application/Json
Salida	• Descripción: Devuelve true si, lo pudo generar con éxito • `{"success":true,"data":true / false}` • `{"success":false,"data":{...error...}}`

```
curl -X POST -H 'Content-Type: application/json' -H 'Authorization:
Bearer xxxxxxxxxx' -i 'http://module_api_url/finalice-firma' --data '{
 "Carpeta": {
    "id": "uuid_carpeta"
  }
}'
```

Tabla 4.3. Definición de la API REST /finalice-firma

4.2.2.2 APIs utilizadas por la aplicación web

Seguidamente, se especifican las APIs del MDT que son llamadas por la aplicación web. En este caso, de acuerdo a la implementación de la plataforma, pueden llegar a omitirse, si fuera el caso en que no se use un módulo de documentos temporales.

<table>
<tr><th colspan="2">POST api_url/guardar-a-firmar</th></tr>
<tr><td>Descripción</td><td>Guarda en el módulo de documentos todos los documentos PDF a firmar asociados a un id de carpeta.</td></tr>
<tr><td>Parámetros de entrada</td><td>Request Body:
Descripción: Recibe el id de carpeta y todos los archivos a firmar en formato Base64.
Formato: Json
Requerido: Sí
Schema:
{‘Carpeta’: {‘id’: String, ‘pdfs’: [{‘id’: String, ‘arch’: String (Base64)}] }}</td></tr>
<tr><td>Authorization</td><td>Bearer xxxxxxxxxx</td></tr>
<tr><td>Headers</td><td>Content-Type: application/json</td></tr>
<tr><td>Salida</td><td>Descripción: la cantidad de archivos guardados con éxito.
Schema: {‘cantidad': Integer}
<code>{"success":true,"data":{"cantidad":1}}</code>
<code>{"success":false,"data":{...error...}}</code></td></tr>
<tr><td colspan="2"><pre>curl -X POST -H 'Content-Type: application/json' -i
'http://module_api_url/guardar-a-firmar' --data '{
 "Carpeta": {
    "id": "uuid_carpeta",
    "pdfs": [{"id":"id1","arch":"base64_1"},
             {"id":"id2","arch":"base64_2"}]
 }
}'</pre></td></tr>
</table>

Tabla 4.4. Definición de la API REST /guardar-a-firmar

<table>
<tr><th colspan="2">POST api_url/recuperar-firmados</th></tr>
<tr><td>Descripción</td><td>Dado un id de carpeta recupera los archivos firmados asociados a ella.</td></tr>
<tr><td>Parámetros de entrada</td><td>Request Body:
Formato: Json
Requerido: Sí
Schema: {‘Carpeta’: {‘id’: String} }</td></tr>
<tr><td>Authorization</td><td>Bearer xxxxxxxxxx</td></tr>
<tr><td>Headers</td><td>Content-Type: application/json</td></tr>
<tr><td>Salida</td><td>Schema: [{‘id': String, ‘arch': String (Base64)}]
<code>{"success":true,"data":[{"id":"id1","arch":"base64_1"}, ...]</code>
<code>{"success":false,"data":{...error...}}</code></td></tr>
<tr><td colspan="2"><pre>curl -X POST -H 'Content-Type: application/json' -H 'Authorization: Bearer xxxxxxxxxx' -i 'http://module_api_url/recuperar-firmados' --data '{
 "Carpeta": {
    "id": "uuid_carpeta"
  }
}'</pre></td></tr>
</table>

Tabla 4.5. Definición de la API REST /recuperar-firmados

<table>
<tr><th colspan="2">POST api_url/es-firma-finalizada</th></tr>
<tr><td>Descripción</td><td>Dado un id de carpeta retorna true si el firmador terminó de firmar y ya dejó a todos los documentos firmados en el MDT, caso contrario retorna false.</td></tr>
<tr><td>Parámetros de entrada</td><td>Request Body:
Formato: Json
Requerido: Sí
Schema: {‘Carpeta’: {‘id’: String} }</td></tr>
<tr><td>Authorization</td><td>Bearer xxxxxxxxxx</td></tr>
<tr><td>Headers</td><td>Content-Type: application/json</td></tr>
<tr><td>Salida</td><td>Schema: true/false
{"success":true,"data":true / false}
{"success":false,"data":{...error...}}</td></tr>
<tr><td colspan="2">curl -X POST -H 'Content-Type: application/json' -H 'Authorization: Bearer xxxxxxxxxx' -i 'http://module_api_url/es-firma-finalizada' --data '{
"Carpeta": {
"id": "uuid_carpeta"
}
}'</td></tr>
</table>

Tabla 4.6. Definición de la API REST /es-firma-finalizada

4.2.2.3 *Secuencia de invocación de las APIs*

A continuación, se muestra un diagrama de secuencia UML[12] que muestra la secuencia de invocaciones a las APIs mencionadas en los puntos 4.2.2.1 y 4.2.2.2. Dicho diagrama, permite describir el orden secuencial de comunicación de los componentes de la plataforma entre sí a través de las APIs.

[12] UML – Lenguaje Unificado de Modelado, es un lenguaje visual para especificar, construir y documentar los artefactos de los sistemas.

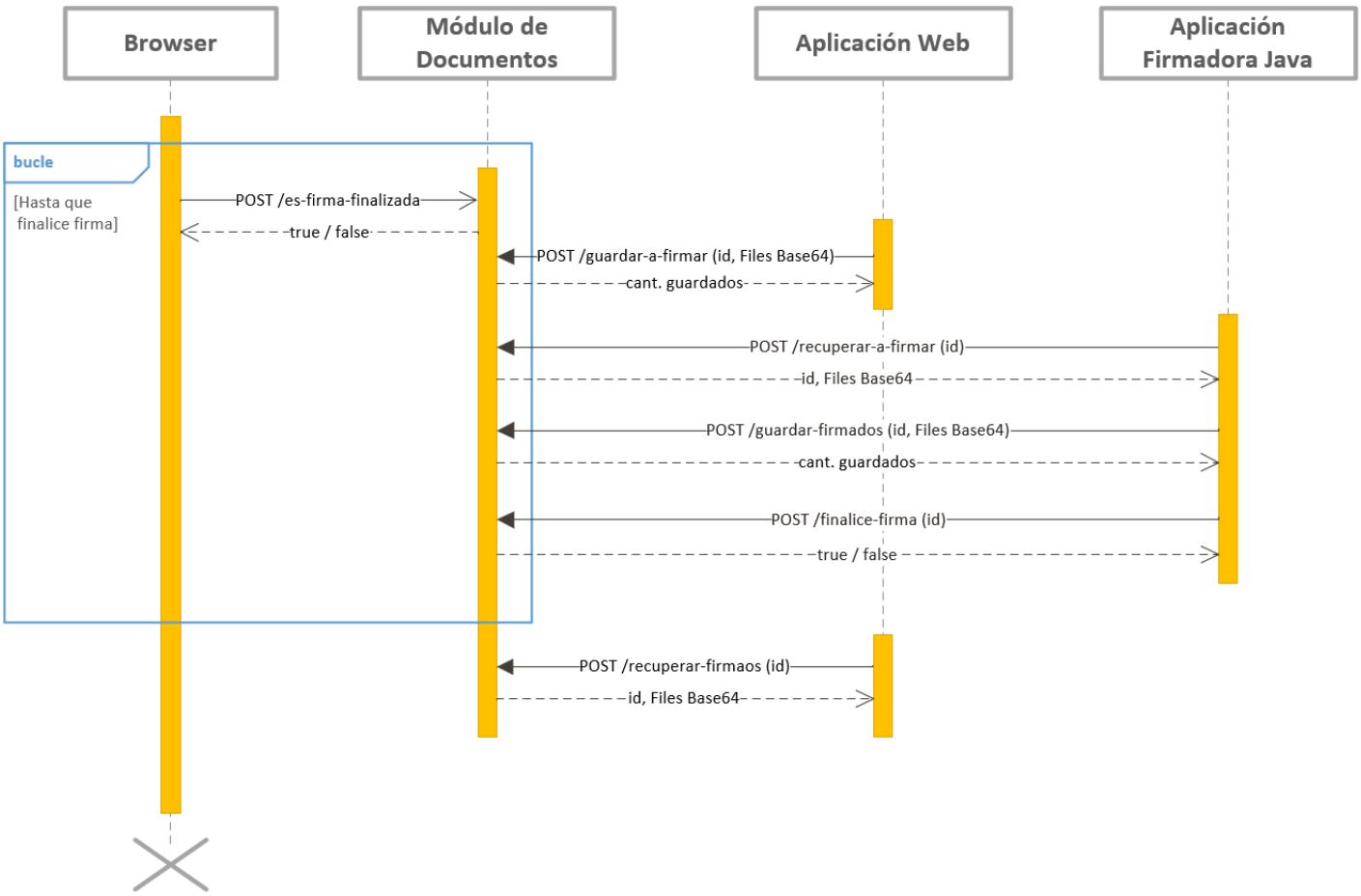

Figura 4.3. Diagrama UML de Secuencia –Invocación a las APIs entre componentes

4.2.3 Aplicación Desktop Java

El firmador es una aplicación de escritorio desarrollada con Java SE Development Kit 8, que corre sobre una Java Virtual Machine v8. La función principal de la aplicación es el firmado de los documentos PDF que fueron seleccionados desde la aplicación web.

4.2.3.1 Funcionamiento

Como se ha mencionado, la aplicación corre en la PC del cliente, por lo que debe estar previamente instalada y configurada en su computadora. De ello resulta que, para iniciar la tarea de firmado es necesario utilizar algún método para que la aplicación desktop si ejecute de manera automática cuando el usuario inicia el proceso desde la aplicación web. Esto se resuelve basándose en la funcionalidad de asociación de archivos que poseen los sistemas operativos. El funcionamiento es el siguiente:

1. El navegador descarga automáticamente un archivo generado por la aplicación, que tiene una extensión específica **.sign**. Este archivo sólo contiene el identificador de tarea (uuid de carpeta) que indica los archivos seleccionados para firmar.
2. Se configura, por una única vez en el sistema operativo, que los archivos con extensión .sign sean abiertos por la aplicación desktop firmadora.
3. Luego, la próxima vez que el usuario firme, la aplicación java se ejecutará automáticamente cuando el sistema abra archivo .sing que se descargó (también de manera automática) del navegador.

Luego, una vez que la aplicación java está corriendo, lee el archivo .sing para recuperar el identificador de carpeta. Seguidamente, le solicita al usuario el PIN/Password de su token USB y recupera los archivos del MDT. Todos los documentos a firmar recuperados se guardan en una carpeta temporal en la PC del cliente. Después, el firmador accede al token para recuperar el certificado del firmante y su clave privada. Si dicha operación fue exitosa, inicia el firmado propiamente dicho de cada uno de los archivos utilizando la clave privada. Al terminar, devuelve los documentos firmados al MDT y le avisa que se completó la tarea. Todo el proceso es automático y transparente para el usuario, sólo debe ingresar su PIN/clave para que inicie la tarea y luego esperar la finalización de toda la operación. A continuación, se muestra un diagrama de flujo que describe funcionamiento mencionado.

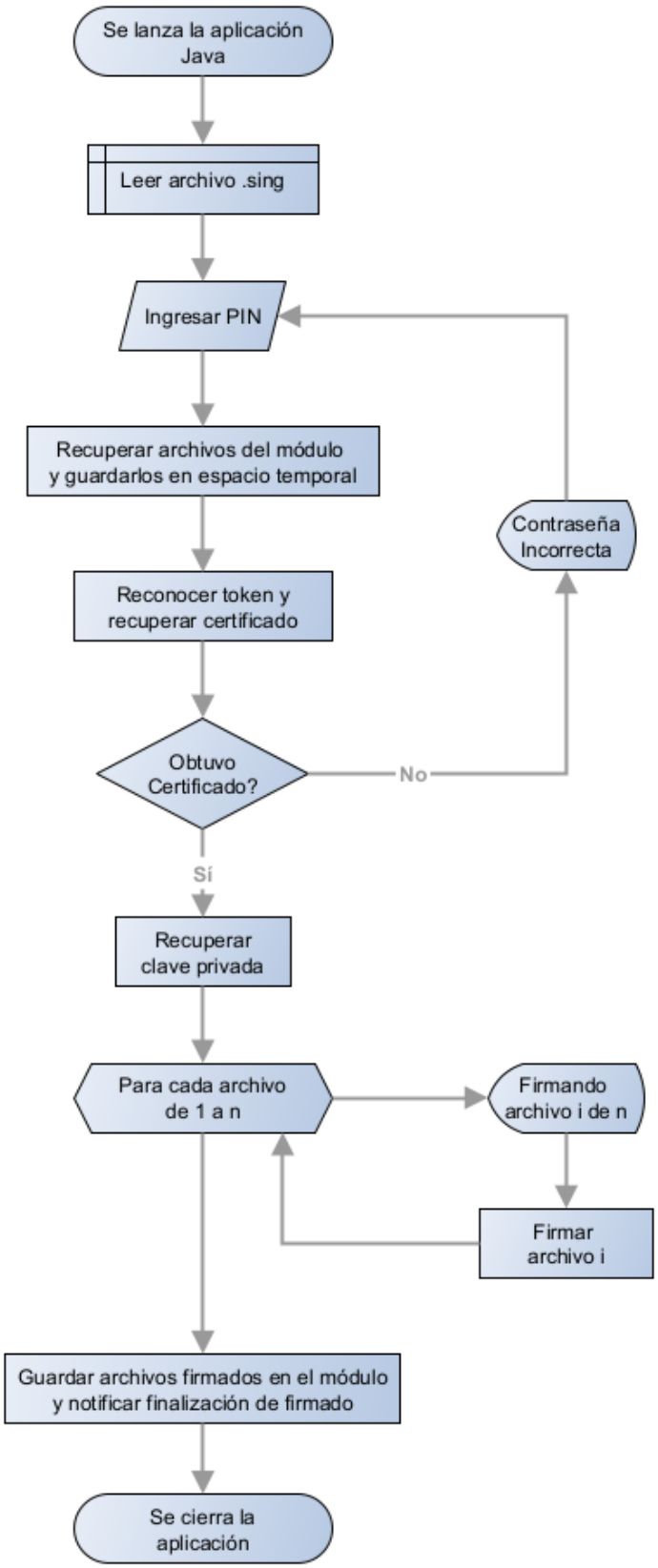

Figura 4.4. Diagrama de Flujo – Aplicación Desktop Firmador Java

4.2.3.2 *Librerías utilizadas*

Para el proceso de firmado propiamente dicho, se utiliza la librería Sun PKCS#11 Provider (sunpkcs11.jar). De acuerdo a la guía de referencia de JDK 8 PKCS#11, dicha librería, en vez implementar los algoritmos criptográficos, actúa de puente entre la API PKCS#11 nativa del dispositivo y las APIs de Java (Java Cryptography Architecture y Java Cryptography Extension, conocidas como JCA y JCE respectivamente). Básicamente, Sun PKCS#11 traduce las llamadas y convenciones entre las dos APIs mencionadas. De esta manera, la librería PKCS#11 nativa, provista por el proveedor del token, tiene que estar presente en el sistema operativo como una librería dinámica (.dll en Windows) o una librería de objetos compartidos (.so en Linux).

Además, para poder administrar el uso de algoritmos criptográficos, se utiliza la librería Bouncy Castle Crypto Provider para JCA y JCE. Del mismo modo, el firmador utiliza la librería de iText 7 Suit que provee una SDK (Software Development Kit) para crear y manipular archivos PDF en Java. Por medio de esta última, es posible el agregado de una estampa al PDF para que queden visible los datos del firmante en el documento.

Por último, para que la aplicación pueda firmar, es necesario disponer del software middleware provisto por el proveedor del token. Este software incluye los drives necesarios para que la computadora detecte automáticamente al dispositivo.

4.3 Modelo Dinámico para el firmado

Partiendo de la estructura estática definida en el diagrama de componentes del punto 4.2, se detalla a continuación un modelo dinámico que muestra la secuencia de pasos que intervienen en tiempo de ejecución durante el firmado desde el momento en que el usuario, desde su navegador, selecciona los documentos a firmar y luego presiona el botón de firmar:

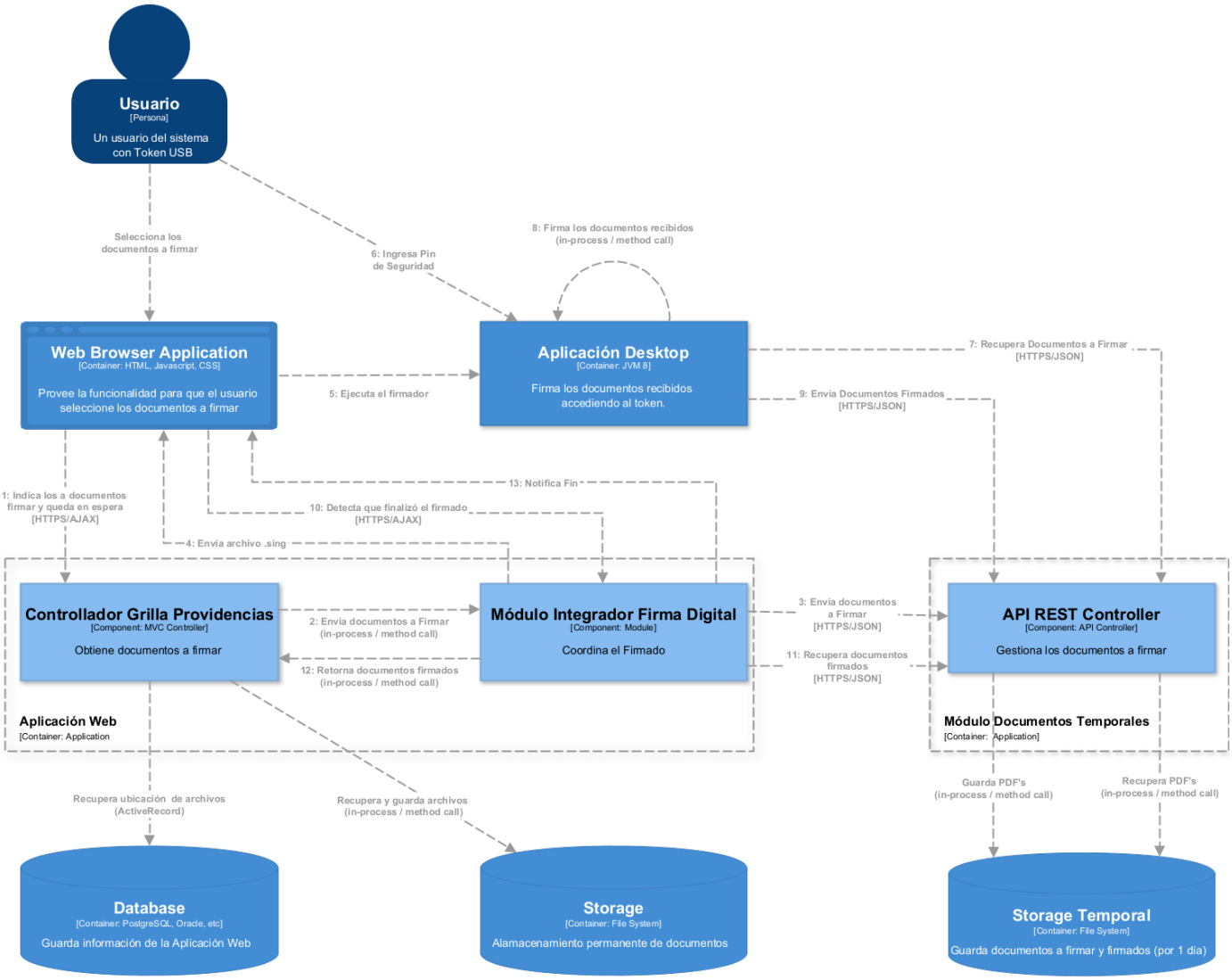

Figura 4.5. Secuencia de firmado – Plataforma de Firma Digital para Aplicaciones Web

A continuación, se explican en detalle los pasos del modelo de la figura anterior, y en los casos que correspondan, se indicará la API invocada:

1. El navegador llama a la aplicación web para iniciar el procedimiento de firma. La llamada es manejada por un controlador específico que recibe además los identificadores de los documentos a firmar. La aplicación web, del lado del cliente, muestra una señal de espera (con un spinner), mientras se queda en un bucle consultando repetidamente al sistema hasta que detecta que finalizó el procedimiento de firmado (API *Tabla 4.6*).

2. El controlador recupera los documentos a firmar del storage y luego inicia al Módulo Integrador de Firma Digital (MIFD) para delegarle el control del proceso.
3. El módulo MIFD integrador genera un identificador de carpeta aleatorio para la tarea de firmado. Luego envía los documentos a firmar al Módulo de Documentos para que los guarde (API *Tabla 4.4*).
4. El módulo MIFD genera un archivo con extensión .sing que contiene el identificador de carpeta (mencionado en el anterior) y genera la descarga del mismo en el navegador del usuario.
5. El navegador, al descargar el archivo con extensión .sing, abre automáticamente la aplicación desktop firmadora Java.
6. El firmador Java solicita el pin/clave al usuario para luego acceder al token.
7. El firmador recupera del módulo de documentos, los archivos a firmar asociados al identificador de carpeta que recuperó el archivo .sing (API *Tabla 4.1*).
8. El firmador accede al token y firma digitalmente todos los documentos PDF.
9. El Firmador envía los documentos PDF firmados al módulo de documentos (API *Tabla 4.2*) y luego le indica que terminó el proceso de firmado (API *Tabla 4.3*).
10. El navegador, que había quedado en espera, consulta una vez más por el estado del firmador al MIFD, y detecta que el firmador terminó con su tarea de firmado (API *Tabla 4.6*).
11. El MIFD recupera los documentos firmados del MDT (API *Tabla 4.5*).
12. El módulo integrador entrega los documentos firmados al controlador de la aplicación web para que los persista.
13. El MIFD notifica al navegador que el firmado terminó exitosamente.

5 Producto

5.1 Fuentes del firmador Java

El proyecto junto con los archivos fuente de la aplicación de desktop Java, están hospedados en el siguiente repositorio de GitHub https://github.com/jusrionegrosistemas/firmadigital. GitHub es una plataforma que ofrece a los desarrolladores, empresas y organizaciones, la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, usando un sistema de

control de versiones. La aplicación subida al repositorio, está estructurada como un proyecto Maven con su respectivo archivo pom.xml (Project Object Model) y todas las dependencias a las librerías necesarias.

5.2 Despliegue de la Plataforma de Firma Digital

5.2.1 Integración de la arquitectura

La arquitectura de la plataforma de firma digital web está compuesta por diferentes módulos como se ha descripto con anterioridad, dicho diseño permite que pueda ser reutilizada e integrada de manera sencilla, en distintos sistemas de la organización u empresa. Sin embargo, a la hora de integrarla en un escenario nuevo, se puede efectuar un análisis previo que determine el costo de su implementación, de acuerdo a las necesidades y recursos disponibles. Si la evaluación arroja un resultado negativo por no ser viable, es posible llevar a cabo una solución alternativa. Se puede reducir el diseño de componentes anteriormente citados, dejando sólo la aplicación web y la aplicación de escritorio firmadora Java. Luego, para que funcione adecuadamente, el único requisito que se debe cumplir, es la implementación de las APIs que consume la aplicación Java definidas en el punto 4.2.2.1. El resto de la lógica se puede implementar en la aplicación web de acuerdo a las necesidades propias. De esta manera la comunicación sólo queda entre la aplicación web, la aplicación desktop y el navegador.

5.2.2 Aplicación de escritorio firmadora

5.2.2.1 Armado de la aplicación

En este apartado se define la estructura que debe tener el artefacto que luego se distribuirá en las computadoras de los usuarios. En primer lugar, necesitamos a la aplicación firmadora. Para ello, se ejecuta “Maven Install”, sobre el proyecto clonado, con lo cual genera la aplicación “firma-0.0.1-SNAPSHOT-jar-with-dependencies.jar”. Luego, dicha aplicación, junto con otros archivos que se encuentran dentro del proyecto, conforman el artefacto final a distribuir.

A continuación, se describe la estructura que debe tener el artefacto:

```
signing.zip:
+---bin
|      firma-0.0.1-SNAPSHOT-jar-with-dependencies.jar
|      OpenArgs.bat
|      OpenArgs.sh
|      +---jre
|            ...(archivos de la JRE 1.8)
+---FirmaConfig
|      config.properties
-
```

5.2.2.2 *Instalación de la aplicación en el cliente*

Para que la aplicación java pueda firmar con un token USB en la computadora del usuario, primeramente, deben instalarse los drivers del dispositivo criptográfico. Generalmente, este software middleware es provisto por la empresa proveedora del token y permite que el sistema detecte automáticamente al token.

Además, para el correcto funcionamiento, es necesario instalar la aplicación desktop firmadora en la computadora del usuario. A continuación, se describen dos tipos de instalaciones que varían de acuerdo al sistema operativo de la PC del usuario final.

5.2.2.2.1 Plataforma Windows

1. Descomprimir el archivo signing.zip en la carpeta home del usuario. Para un usuario cuyo nombre es “miusuario” quedaría la siguiente estructura de carpetas:
    - C:\Users\miusuario**bin**
    - C:\Users\miusuario**FirmaConfig**
2. Editar el archivo C:\Users\miusuario**bin\OpenArgs.bat** de la siguiente manera:
    - CD C:\Users\napellido\bin
    - start /MIN C:\Users\napellido\bin\jre\bin\java.exe -jar firma-0.0.1-SNAPSHOT-jar-with-dependencies.jar %1
3. Editar el archivo C:\Users\miusuario**FirmaConfig\config.properties** y ajustar la configuración de los siguientes parámetros, respetando las dos barras para el separador de carpera como se indica a continuación:
    - **temp**: representa la ubicación de los archivos temporales del usuario actual. Desde una consola se puede obtener con el comando %TEMP%. Por ejemplo: **temp=C:\\Users\\miusuario \\AppData\\Local\\Temp**
    - **url:** representa la ruta al módulo de archivos temporales (servidor de archivos). Por ejemplo: **url=localhost**

- **libreria**: el path a la librería del driver del token, usualmente provista por el proveedor del token. Por ejemplo: **libreria=C:\\Windows\\System32\\asepkcs.dll**

4. Para asociar la aplicación de escritorio firmadora con los archivos de extensión .sign, deberá hacer lo siguiente sólo la primera vez. Primero, desde un navegador, ingresar a la aplicación web, luego durante el proceso de firmado, cuando se descarga el archivo .sign:

    - Elegir abrirlo con el programa **C:\Users\miusuario \bin\OpenArgs.bat**

    - **OpenArgs.bat** debe quedar seleccionado como programa predeterminado para futuras aperturas.

5.2.2.2.2 Plataforma Linux

1. Descomprimir el archivo signing.zip en la carpeta home del usuario. Para un usuario cuyo nombre es “miusuario” quedaría la siguiente estructura de carpetas:
    - /home/misuario**/bin**
    - /home/misuario**/FirmaConfig**
2. Editar el archivo /home/misuario/**bin/OpenArgs.sh** y cambiar en los paths el nombre de usuario por el que corresponda.
3. Editar el archivo /home/misuario/**FirmaConfig/config.properties** y ajustar la configuración de los siguientes parámetros:
    - **temp**: representa la ubicación de los archivos temporales del usuario actual. Por ejemplo: **temp=/tmp**
    - **url:** representa la ruta al módulo de archivos temporales (servidor de archivos). Por ejemplo: **url=localhost**
    - **libreria**: el path a la librería del driver del token, usualmente provista por el proveedor del token. Por ejemplo: **libreria=/usr/lib/x64-athena/libASEP11.so**
4. Copiar **application-x-sign.xml** en **/home/miusuario/.local/share/mime/packages**
5. Actualizar los tipos mime ejecutando desde consola: $ **update-mime-database ~/.local/share/mime**
6. Editar el archivo **openargs.desktop** para cambiar el path, poner el nombre de usuario correspondiente.
7. Copiar el archivo **openargs.desktop** a **/usr/share/applications**
8. Actualizar la lista de aplicaciones ejecutando desde consola: $ **update-desktop-database /usr/share/applications**
9. El firmador funciona con Java 8, para ello se debe consultar por consola con "**java -version**", luego si las versiones son distintas, es necesario instalar la 8:
    - **sudo apt install openjdk-8-jdk**

- Para cambiar a la versión 8: **sudo update-alternatives --config java**
- Si no desea cambiar la versión en el paso anterior, definir el path en el archivo OpenArgs.sh de manera que apunte a la versión de java 8.

10. Para asociar la aplicación de escritorio firmadora con los archivos de extensión .sign, deberá hacer lo siguiente sólo la primera vez. Primero, desde un navegador, ingresar a la aplicación web, luego durante el proceso de firmado, cuando se descarga el archivo .sign:
    - Desde Mozilla Firefox: seleccionar abrir siempre con la aplicación OpenArgs.
    - Desde Chrome: hacer click derecho y seleccionar abrir siempre ese tipo de archivo con OpenArgs.

6 Caso de éxito

La Plataforma de Firma Digital para Aplicaciones Web se integró con éxito a la aplicación web Sistema de Gestión de Expedientes Judiciales (PUMA) y comenzó a utilizarse por empleados, secretarios y jueces (de los fueros Laboral y Originario) del Poder Judicial de Río Negro en marzo del año 2021, junto con la puesta en producción de la aplicación web mencionada. Actualmente, se firman aproximadamente 1200 actos procesales diarios en toda la provincia.

La aplicación web y el módulo de documentos temporales fueron desarrollados con el framework Yii2 de PHP. Adicionalmente, el módulo integrador de firma digital se implementó como un widget en Yii2, para poder reutilizarlo en futuros sistemas que necesiten firmar digitalmente.

Los tokens usb utilizados son MS-IDProtect de la empresa Macroseguridad, están preparados para funcionar con PKI, son compatibles con PKCS#11 y están homologados por la ONTI. Las librerías utilizadas para el acceso al dispositivo y firmado son las provistas por el proveedor de los tokens: asepkcs.dll (para sistemas Windows) o libASEP11.so (para sistemas Linux).

A continuación, se detalla un ejemplo con capturas de pantalla, en donde un empleado, firma cuatro documentos de diferentes actos procesales de un expediente judicial. En primer lugar, el usuario accede a una grilla de actos procesales (movimientos) y filtra por aquellos que están en estado pendiente de firma. Luego,

selecciona los que desea firmar y presiona el botón de firmado, remarcado por un círculo rojo.

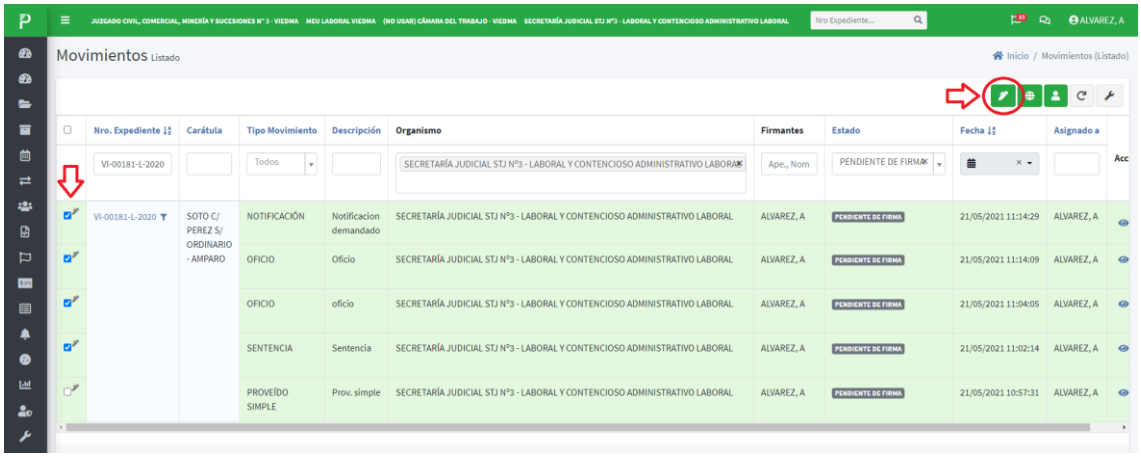

Figura 6.1. El empleado selecciona los actos procesales a firmar desde la grilla de movimientos.

Una vez accionado el botón de inicio de firma, el sistema le pide al empleado que ingrese el PIN de seguridad del token USB que debe tener conectado a su computadora. Como se muestra a continuación.

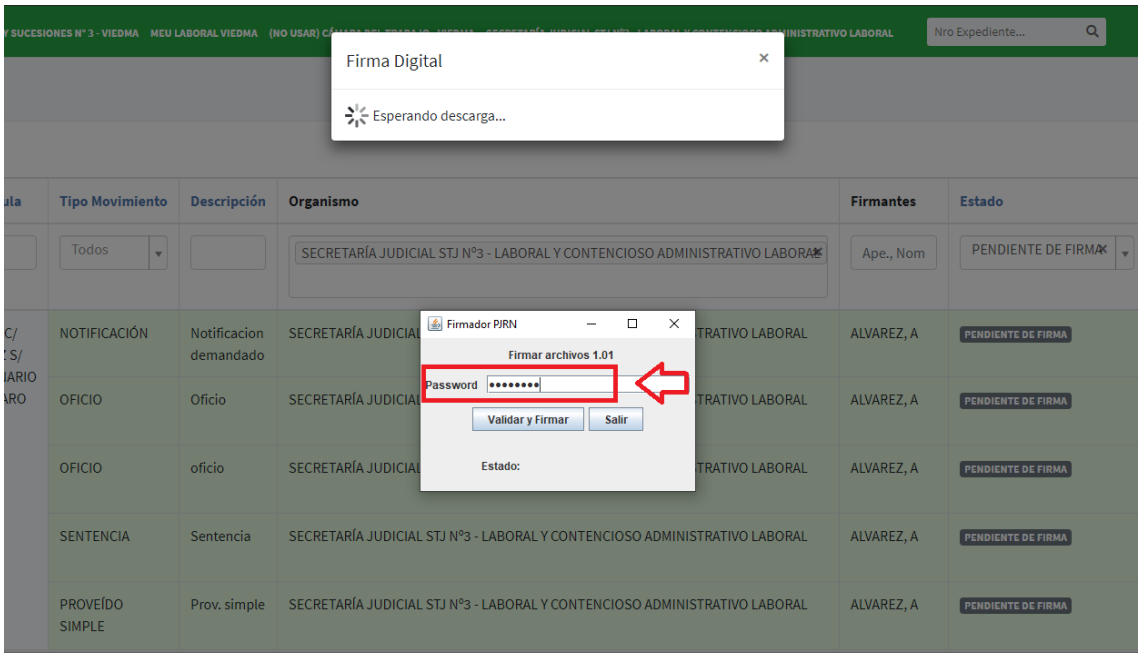

Figura 6.2. El empleado ingresa su PIN de seguridad

Luego, el sistema recupera los archivos a firmar, accede al token y comienza a firmarlos digitalmente de a uno.

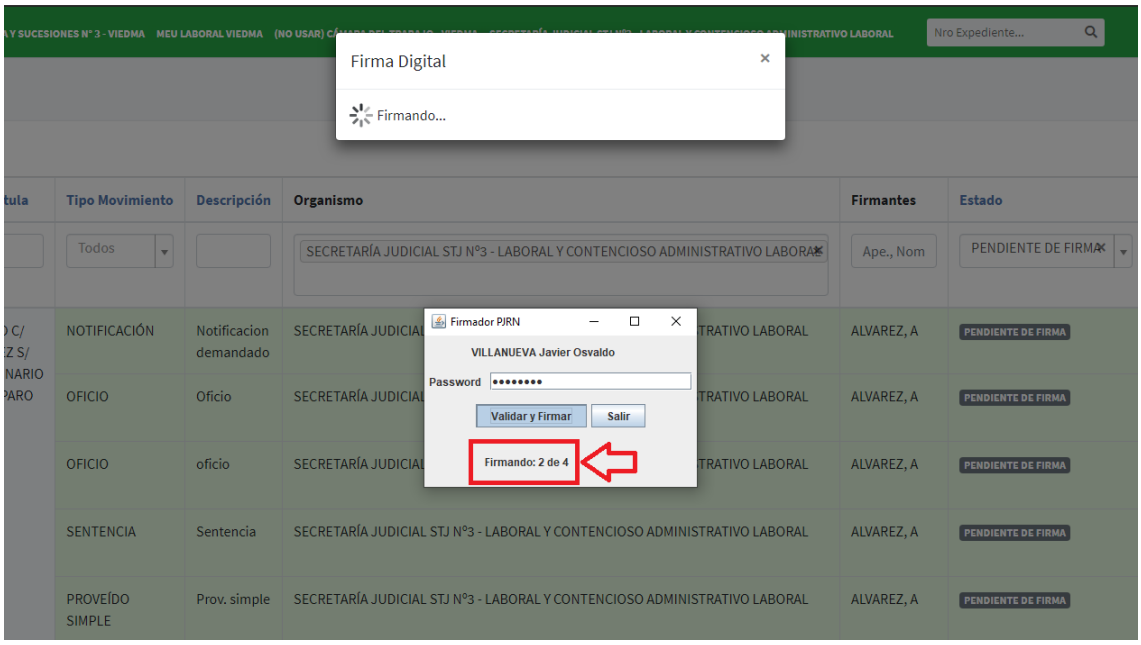

Figura 6.3. La ampliación desktop inicia el firmado de los cuatro documentos.

Finalmente, cuando el sistema de gestión de expedientes recuperó todos los documentos ya firmados, si todo salió bien, informa que finalizó con éxito. Por cierto, los documentos de los actos procesales ya firmadas por el empleado, pueden ser firmados por otros funcionarios y/o jueces. Es decir, al firmar un documento, éste no pierde las firmas previas.

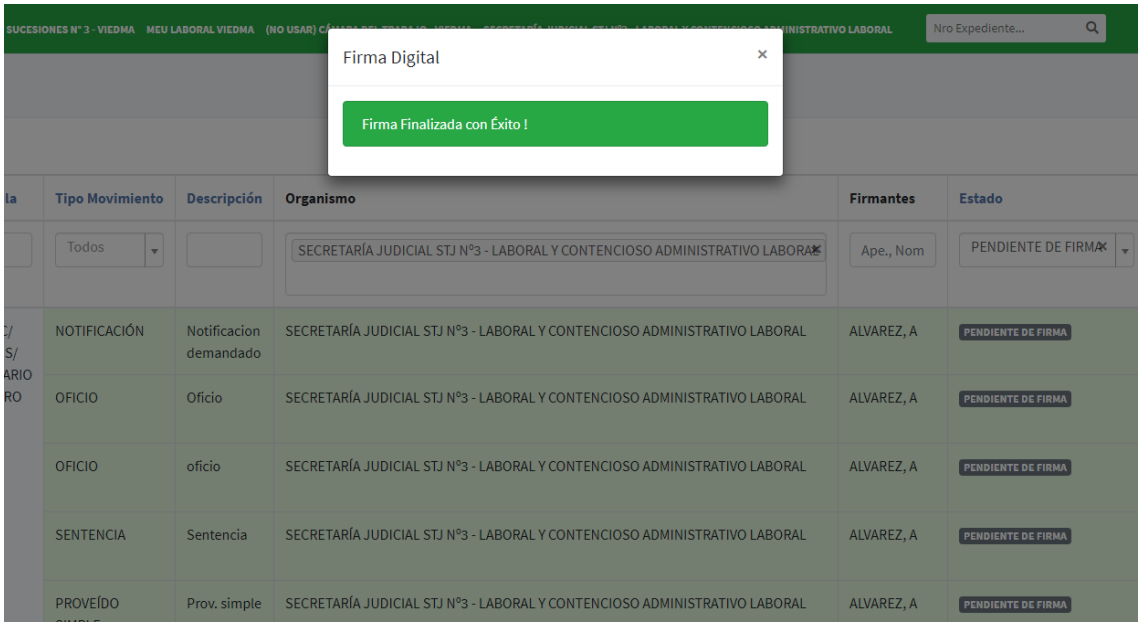

Figura 6.4. El sistema informa al usuario que el firmado finalizó con éxito.

7 Conclusiones y líneas futuras

7.1 Conclusiones

La tarea de firmar digitalmente desde una aplicación web con un dispositivo criptográfico no suele resultar una experiencia muy amigable para el usuario. En este trajo final se revisaron las alternativas existentes y, si bien algunas se acercan en mayor o menor grado a resolver la necesidad de firmar de manera transparente múltiples documentos con un token, todas presentan alguna desventaja.

Adicionalmente, se analizó el motivo por el cual no es posible acceder al token del usuario desde su navegador web. Los browsers modernos renderizan las páginas web dentro de sandboxes que impiden acceder a los recursos y dispositivos del sistema por motivos de seguridad, protegiendo al usuario de posibles riesgos y ataques.

A partir de los análisis anteriores, se mostró el diseño y la arquitectura del la Plataforma de Firma Digital para Aplicaciones Web necesaria para resolver el problema. Del mismo modo, se mostró la implementación de una aplicación desktop firmadora utilizada por la plataforma de firmado, que logra firmar del lado del cliente, documentos PDF con un token USB utilizando el estándar PKCS#11.

La solución descripta cumple con los objetivos propuestos, simplifica la tarea de firmado para el usuario a unos pocos clics y permite firmar de a varios documentos PDF a la vez. Además, se agrega en el PDF firmado, una estampa visible en el documento con los datos del firmante. Del mismo modo, la plataforma de firma digital fue diseñada con el objetivo de poder reutilizarla e integrarla desde cualquier aplicación web de la empresa u organismo en donde se implemente.

Finalmente, se concluye mostrando el caso de éxito del Poder Judicial de Río Negro, donde actualmente está siendo usada. Actualmente, gestiona en su totalidad los expedientes judiciales correspondientes a dos fueros, alcanzando a firmar cerca de 1200 actos procesales diarios, utilizando la solución descripta.

7.2 Líneas Futuras

Si bien, como se ha mencionado, la plataforma de firmado se encuentra actualmente operativa en el ambiente productivo, hay ciertos puntos que pueden mejorarse. Respecto a la tarea de instalación del aplicativo de escritorio Java, actualmente es manual y podría agilizarse utilizando algún tipo software empaquetador.

En referencia al firmador Java, sería adecuado agregar la posibilidad de poder probar que el certificado no estaba revocado por la entidad emisora (AC) al momento que el usuario firmó el documento. Esto puede llevarse a cabo de dos maneras, embebiendo en el PDF la lista de certificados revocados (CRL) de la Autoridad Certificante o especificando una URL que se encarga de chequear el estado de revocación del certificado, esto es, a través del protocolo OCSP (Online Certificate Status Protocol).

Por otro lado, es aconsejable agregar un estampado de tiempo (TimeStamp) al documento al momento de firmar, con el fin de asegurarnos que el certificado no estaba revocado o expirado a la hora del firmado. Actualmente el firmador, toma como fecha de firmado la del reloj de la computadora personal del usuario y esto no es del todo seguro. Este problema se resuelve en el firmado invocando a una tercera parte, que consta de un servicio en línea llamado Autoridad de Estampado de Tiempo (Time Stamping Authority, TSA, inglés).

Adicionalmente, para poder verificar la validez del documento firmado en un futuro lejano, se debe incluir al estándar LTV (Long Term Validation). Al agregar este estándar, se garantiza que los documentos firmados digitalmente contengan todos los elementos necesarios que permiten validar la firma durante un largo período de tiempo. Esto es posible, debido a que añade información extra al documento como estampado de tiempo a nivel de documento, cadenas de certificados, información de revocación y expiración, e información de los algoritmos criptográficos utilizados.

8 Referencias

1. Ferri, L., Pichetti, L., Secchi, M., y Secomandi, A. (2010). U.S. Patent Application No. 12/359,457. Sandbox Web Navigation. Recuperado de https://bit.ly/31yaErh [Consultado noviembre 2021]
2. XolidoSign Homae Page, Recuperado de https://bit.ly/3yFbL4G [Consultado noviembre 2021]
3. SIU-Toba Firmador, Recuperado de https://bit.ly/3oL35VA [Consultado noviembre 2021]
4. Oracle. (2016). Migrating from Java Applets to plugin-free Java technologies. Recuperado de https://bit.ly/2ZSYmZy [Consultado noviembre 2021]
5. Oracle. (2020). Java Client Roadmap Update. Recuperado de https://bit.ly/3mM4RGG [Consultado noviembre 2021]
6. Plataforma de Firma Digital Remota. Recuperado de https://bit.ly/3oYiPVD [Consultado noviembre 2021]
7. Simon Brown, C4 Model Home Page. Recuperado de https://bit.ly/3EJ18jk [Consultado noviembre 2021]
8. iText7, Addendum to Digital signatures for PDF documents, Recuperado e https://bit.ly/3bRgxlP [Consultado noviembre 2021]
9. Lucena López, M. J. (2021). Criptografía y Seguridad en Computadoras. Versión 5-1.1.1. Recuperado de https://bit.ly/3BLCRak [Consultado noviembre 2021]
10. Wadhwa, N., Hussain, S. y Rizvi, S. (2013). A Combined Method for Confidentiality, Integrity, Availability and Authentication (CMCIAA). Proceedings of the World Congress on Engineering, 1-4. Recuperado de https://bit.ly/3k687Lq [Consultado noviembre 2021]
11. Medina, Y. y Miranda, H. (2015). Comparación de algoritmos basados en la criptografía simétrica DES, AES y 3DES. Mundo FESC, 1(9), 14-21. Recuperado de https://bit.ly/3weOiGF [Consultado noviembre 2021]
12. Albarqi, A., Alzaid, E., Ghamdi, F., Asiri, S. y Kar, J. (2015). Public Key Infrastructure: A Survey. Journal of Information Security, 6(1), 31-37. https://bit.ly/3GSCvT1 [Consultado noviembre 2021]

13. Jefatura de Gabinete de Ministros. Ente Licenciante. Recuperado de https://bit.ly/3q4DMAv [Consultado noviembre 2021]

14. OASIS Standard. (2015). PKCS #11 Cryptographic Token Interface Base Specification Version 2.40. Recuperado de https://bit.ly/3EJ18jk [Consultado noviembre 2021]

15. Yongge W. (2012). Public-Key Cryptography Standards: PKCS, University of North Carolina at Charlotte. Recuperado de https://bit.ly/3o11Dyq [Consultado noviembre 2021]

16. ETSI TS 102 778-1. (2009). (European Telecommunications Standards Institute), Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES, Recuperado de https://bit.ly/3GR5nLK [Consultado noviembre 2021]

17. Reis, C., Barth, A., y Pizano, C. (2009). Browser Security: Lessons from Google Chrome. ACM Queue. Recuperado de https://bit.ly/3vZuJRu [Consultado noviembre 2021]

18. Bhavaraju, S, Smith, T. y Zhang, B. (2018). Security Analysis of Firefox WebExtensions. Computer and Network Security. Recuperado de https://bit.ly/3vVyJTl [Consultado noviembre 2021]

19. Lowagie B. (2012). Digital Signatures for PDF Documents. Recuperado de https://bit.ly/3wgLsAQ [Consultado noviembre 2021]

20. Oracle. JDK 8 PKCS#11 Reference Guide. Recuperado de https://bit.ly/3k84LHB [Consultado noviembre 2021]

21. The Legion of the Bouncy Castle Home Page. Recuperado de https://bit.ly/2ZWfv4w [Consultado noviembre 2021]

22. Villanueva J. y Molinari E. (2021). Plataforma de Firma Digital para Aplicaciones Web del Poder Judicial de Río Negro. SIE 2021, Simposio de Informática en el Estado. 50º Jornadas Argentinas de Informática e Investigación Operativa (JAIIO). Estará disponible en breve desde https://bit.ly/3k8XI1u [Consultado noviembre 2021]

23. MacroSeguridad Home Page. Recuperado de https://bit.ly/3wpfBy4 [Consultado noviembre 2021]

24. Adobe Acrobat Home Page. ¿Qué es el formato PDF?. Recuperado de https://adobe.ly/3BYnzzr [Consultado noviembre 2021]

25. ISO 32000 (PDF) – PDF Association. Recuperado de https://bit.ly/3CMz0v9 [Consultado noviembre 2021]

26. InfoLEG, Información Legislativa. Ley Nº 25.506. Recuperado de https://bit.ly/3CTCvA0 [Consultado noviembre 2021]