PROYECTO FINAL

Implementación de un Sistema de Firma Electrónica Interno con Encriptación de Curva Elíptica

Nicolás Valenzuela

Ingeniería Informática
Facultad de Ingeniería
Universidad Católica de Salta
2017

Título del trabajo:

Implementación de un Sistema de Firma Electrónica Interno con Encriptación de Curva Elíptica

Alumno: Nicolás Valenzuela _____

Profesor Tutor: Lic. Fredi Rene Aprile _____

Tribunal Evaluador:

Lic. Gisela Rottigni _____

Ing. Guillermina Nieva _____

Ing. Mauricio Guaymás _____

Fecha de Exposición del Trabajo:

A mis padres y hermana, que siempre me apoyaron en todo.
A mis amigos Nacho y Tito, con los que compartí este hermoso viaje que fue la universidad.

Abstract

En el presente Proyecto de Grado se propone plantear la creacíon de un Sistema Interno de Firma Electrónica empleando Criptografía de Curva Elíptica, destinado a mejorar una problemática real. Se exploraron los fundamentos para iniciar el proyecto y para la elección del esquema criptográfico, las metodologías empleadas y se detalló el desarrollo del mismo. Finalmente, fue implementado de manera exitosa un prototipo, produciendo un sistema base apto para su utilización en un ambiente de trabajo real.

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Capítulo 1

Introducción

La idea de implementar un Sistema de Firma Electrónica Interno surgió durante mis experiencias trabajando como Analista Desarrollador pasante. Durante la participación en proyectos de desarrollo de software que involucran un considerable número de personas, son frecuentes los intercambios de archivos a través de diferentes sistemas informáticos de comunicaciones, tales como correos electrónicos, software de mensajería, o el mismo sistema de archivos compartidos en redes de Microsoft Windows. En ocasiones, un grupo de trabajo puede utilizar más de un sistema de comunicaciones para intercambiar archivos. Al tratarse de medios digitales de telecomunicaciones, estos son propensos a recibir diferentes tipos de ataques que vulneran la integridad de la información que se está transmitiendo.

La firma digital es un concepto informático fundamental para garantizar la autenticidad e integridad de la información que se está transmitiendo a través de medios digitales de comunicación, y es una solución a la problemática previamente mencionada. Ciertos sistemas de comunicaciones, tales como los protocolos de correo electrónico, poseen herramientas embebidas para generar firmas digitales de los correos que los usuarios envían, y es un método ampliamente utilizado en entidades gubernamentales de Argentina actualmente. Otros sistemas, como ciertos software de mensajería, no poseen métodos de generación de firmas digitales, y presentan un riesgo a la hora de transmitir información de forma segura.

Si una organización requiere que sus miembros realicen intercambios de archivos utilizando firma digital, implicaría un esfuerzo de capacitación y entrenamiento en temáticas como utilización y manipulación de claves pública y privada, y para la utilización de alguna de las vastas herramientas de generación de firmas digitales que existen actualmente en el mercado.

En el presente Proyecto de Grado, se analizarán en profundidad las problemáticas previamente mencionadas, y se detallará la solución propuesta, que es la implementación de un Sistema de Firma Electrónica Interno utilizando Encriptación de Curva Elíptica. Se analizará la elección del esquema criptográfico de Curva Elíptica, y las diversas tecnologías utilizadas para conseguir con éxito el abordaje de una solución.

Es de vital importancia aclarar que el concepto informático utilizado es el de “Firma Digital”, el cual es un elemento sustancial en la temática de este proyecto de grado. La legislación argentina actual especifica dos denominaciones para este instrumento informático: “Firma Digital” y “Firma Electrónica”, diferenciando entre ambos el valor probatorio que tienen ante la ley. Tal como se especifica en los Artículos 2° y 5° de la Ley 25.506 de Firma Digital [1], una firma digital posee una presunción “iuris tantum” a su favor. Es decir, un documento firmado digitalmente es automáticamente verificado como correcto, salvo que el demandante pruebe lo contrario. Por otra parte, en una firma electrónica se invierte la carga probatoria: corresponde a quien invoca su autenticidad acreditar su validez. Detallando las cuestiones técnicas de estas denominaciones, en

[1] Ley N° 25.506 de Firma Digital, sancionada por el Honorable Congreso de la Nación Argentina el 14 de Noviembre de 2001

el marco legal argentino denominamos “firma digital” a las que hayan sido generadas utilizando un certificado dentro de la “chain of trust” (cadena o jerarquía de confianza) de la Autoridad Certificante Raíz de la República Argentina. Por otra parte, las firmas generadas utilizando un certificado de cualquier otra Autoridad de Certificación que no esté contemplada bajo la “AC Raíz” de la República Argentina, es denominada “firma electrónica”. A modo de ejemplo, en la figura 1.1 podemos observar el certificado de la Autoridad Certificante de la AFIP, que es una de las Autoridades Certificantes pertenecientes a la ruta de certificación de la “AC Raíz” de la República Argentina. Todas las firmas generadas empleando claves con certificados firmados por la Autoridad Certificante de la AFIP serán generadas bajo la ruta de certificación de la “AC Raíz” y por lo tanto, se encuentran contempladas bajo la legislación de Firma Digital.

Debido a que este proyecto pretende implementar una solución que sea utilizada tanto en ámbito público como privado, para ser utilizado en cualquier Infraestructura de Clave Pública existente o por crearse, la denominación que utilizaremos en todo este informe será “Firma Electrónica”.

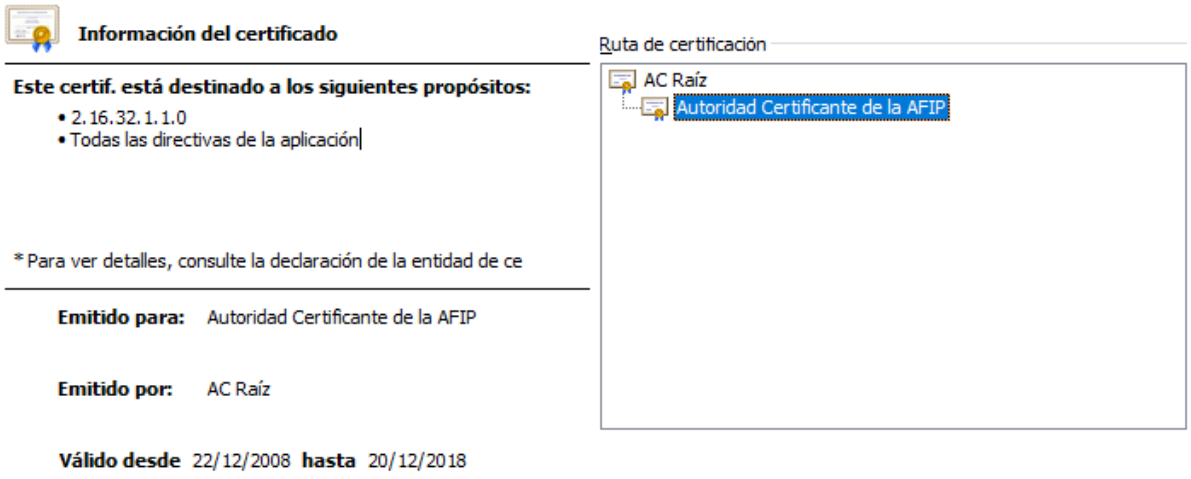

Figura 1.1: Ejemplo de Certificado de una Autoridad Certificante en Argentina: AFIP

Capítulo 2

Estado de la cuestión

La Firma Electrónica es una herramienta tecnológica indispensable a la hora de garantizar autoría e integridad de documentos digitales en ámbitos donde es indispensable que las comunicaciones se realicen de manera segura.

Tal como se mencionó en la introducción de este proyecto de grado, existen sistemas de comunicaciones, tales como los de correo electrónico, que poseen protocolos específicos para generación y transmisión de firmas electrónicas de los mensajes transmitidos. A su vez existen formatos de archivos, tales como los archivos Portable Document File (.pdf), o los Documentos de Microsoft (.docx), cuyos software de edición incluyen herramientas para incluir una firma electrónica en el mismo archivo.

Por otra parte existen otros tipos de archivo, como ser archivos de imágenes, de sonido o de video, que no presentan un software estándar de firma electrónica. Esto implica la utilización de diferentes herramientas para efectuar la tarea, haciendo que las operaciones de firmado y de comprobación sean dificultosas al tener que recurrir a una gran variedad de software según el tipo de archivo con el que se esté tratando.

Antes de abordar el análisis del problema y de la solución propuesta, repasaremos las problemáticas más comunes que motivan la utilización de firmas electrónicas, conceptos de criptografía, esquemas criptográficos e infraestructuras de clave pública.

2.1. La problemática de los documentos digitales

Hankerson, Menezes y Vanstone, en su libro *Guide to Elliptic Curve Cryptography*[1] detallan adecuadamente las problemáticas encontradas en el intercambio de documentos digitales:

Plantearemos un modelo básico de comunicaciones digitales: Un usuario “A” que envía un documento digital a un usuario “B” a través de cualquier medio. En un mundo ideal el documento llegaría a su destino sin que ningún intermediario lo modifique. Además, el usuario “B” estaría seguro que el documento fue redactado por el usuario “A” y no por un impostor.

En el mundo real no todas las personas actúan de manera honesta, y es muy probable que exista alguien que quiera obrar contra los intereses de los usuarios “A” y “B”.

En un modelo de redes o sistemas informáticos que no utilice algún método de certificación en la información que se transmite, la falta de un elemento que garantice la veracidad de esa información hace que podamos identificar fácilmente las siguientes problemáticas:

- **La autoría de los documentos transmitidos no es determinable con certeza:** Mediante las herramientas adecuadas, un usuario puede enviar un documento a través de una red

[1] Darrel Hankerson, Alfred J. Menezes y Scott Vanstone. 2010. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Inc. ISBN: 0-387-95273-X.

afirmando ser otra persona. El receptor no tendrá ninguna manera de determinar si el autor del documento es realmente el que dice ser, o un intruso.

En la figura 2.1 , el usuario “A” envía un documento a “B” simulando ser “A”. “B” no podrá saber con certeza si el documento realmente fue creado por “A”.

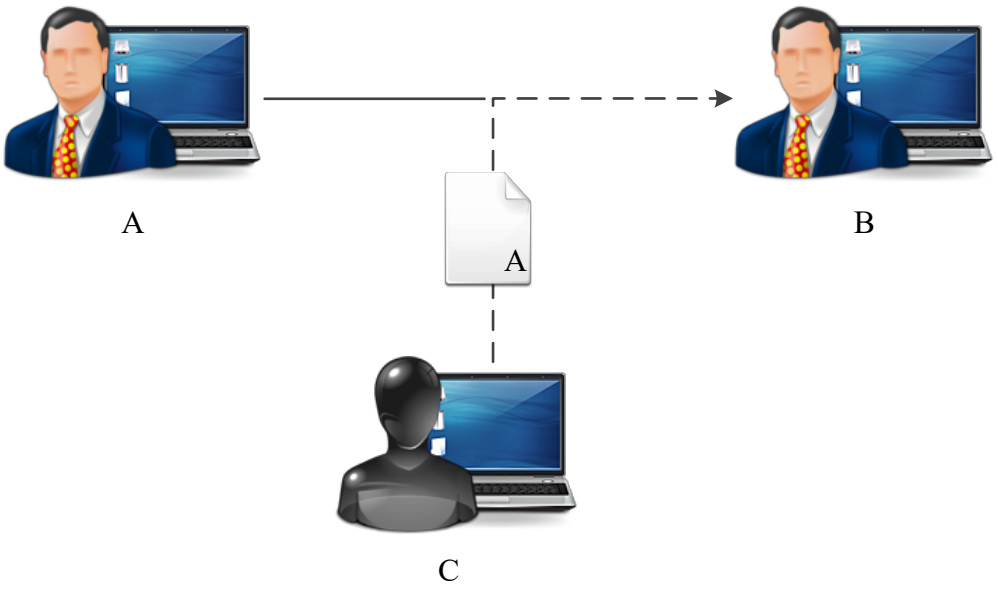

Figura 2.1

- **Un documento electrónico puede ser alterado fácilmente, sin detección:** En un documento físico, ya sea impreso o manuscrito, las modificaciones que este tenga son relativamente fáciles de detectar. Si encontramos una palabra tachada, zonas con corrector líquido o algún daño en el papel, tenemos indicios de que el documento fue alterado y muy probablemente la información que contenga actualmente no coincida con la que contenía originalmente.

  En un documento digital la información contenida puede ser modificada con las herramientas adecuadas sin que los cambios sean detectados. Podemos ejemplificar lo anterior en la figura 2.2. “A” envía un documento a “B”, pero este es interceptado por “A” antes de llegar a destino modifica el documento y lo envía a “B”. Este último pensará que el documento no está alterado y que la información que contiene coincide con lo que envió “A” originalmente. Si “A” no revisa el documento que “B” recibió, esta alteración pasará desapercibida y no habrá forma de verificarlo.

  Por otra parte, en la figura 2.3 podemos encontrar el caso en donde “B” recibe correctamente el documento de “A” (1), pero “A” accede a la computadora que utiliza el usuario “B” y lo modifica sin que este último lo perciba (2).

- **Un documento digital es repudiable:** Un usuario puede crear un documento digital, distribuirlo y luego negar su autoría. Como el mismo documento puede ser creado exactamente igual en diferentes computadoras y por diferentes usuarios, entonces su autor original puede negar haber creado un determinado documento, sin que los demás tengan forma alguna de comprobarlo.

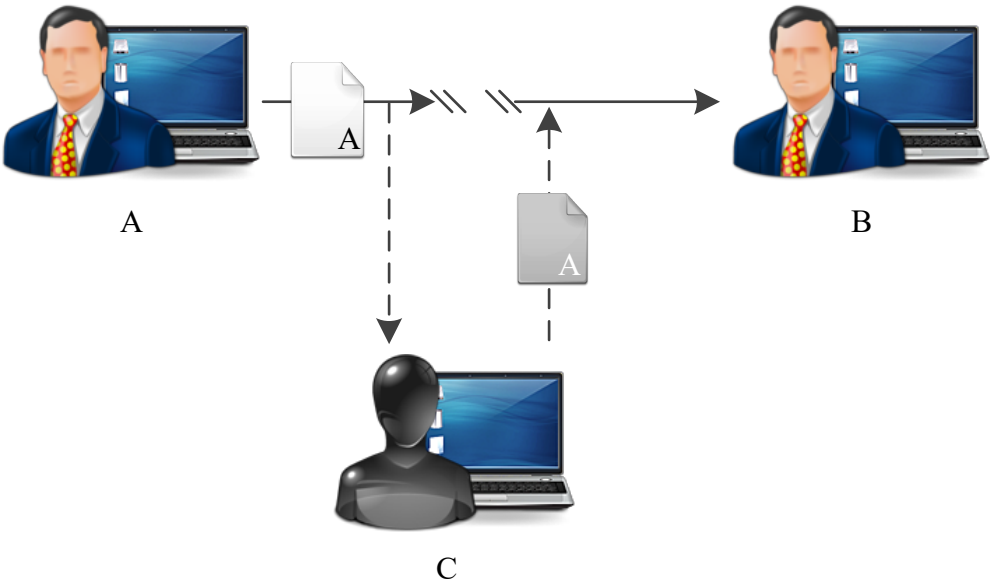

Figura 2.2

2.1.1. Solución mediante una firma digital

El concepto de "firma digital" surge a partir de la necesidad de brindar confianza, autenticidad y seguridad en las transacciones que se realizan digitalmente, a través de computadoras.

Una firma digital es elemento informático que nos permite validar el autor de un documento, y autenticar la integridad del contenido. Mediante una firma digital proveemos soluciones a las problemáticas mencionadas antes:

- **Autoría determinable a través de una firma digital:** Mediante una firma digital podemos identificar quién creó el documento que recibimos, en forma fehaciente. Un impostor que quiera enviarnos un documento firmado simulando ser otra persona no podrá hacerlo, ya que no tiene la información necesaria para generar la firma digital. La firma de un usuario solo puede ser generada por ese usuario.
- **Integridad del documento firmado:** En un documento firmado se puede verificar que el contenido coincide con la firma digital. Si ese documento fue alterado luego del firmado, la firma digital no coincidirá.
- **No-repudio del documento firmado:** Un usuario no puede negar haber firmado un documento cuando sí lo hizo previamente. Las firmas digitales son generadas con información que solo el firmante conoce. De esta manera, una firma digital no puede ser repudiada.

2.1.2. Composición y funcionamiento

Una firma digital debe funcionar y estar compuesta de forma que garantice las soluciones previamente mencionadas. Cada una de estas soluciones se consigue de la siguiente manera:

2.1.2.1. Conseguir integridad del documento firmado

Una firma digital debe contener un elemento que nos indique si el documento firmado contiene exactamente la información para la cual la firma fue generada. Cuando comparamos un documento con su firma digital, debemos poder saber si el documento fue alterado o no. Este elemento de comparación es el "hash", también llamado función resumen o función digesto.

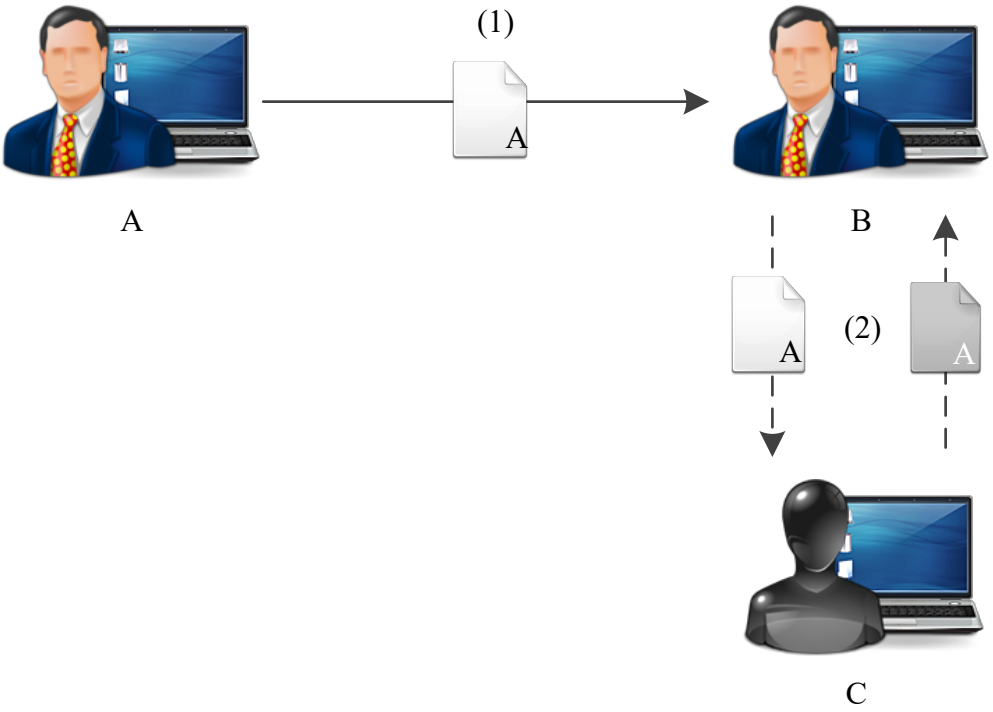

Figura 2.3

Función Hash Una función Hash es un algoritmo que tiene como entrada un conjunto de elementos, que suelen ser cadenas de caracteres, y los convierte en una cadena de salida de longitud finita y fija. El objetivo básico de un valor "hash", obtenido a partir de una cadena de caracteres, es que sirva como una representación compacta de dicha cadena. Por esta razón, si ingresamos en una función Hash dos cadenas de caracteres con alguna diferencia entre ellas, por más mínima que sea, producirá dos valores "hash" totalmente diferentes. Ejemplo de funcionamiento de una función Hash:

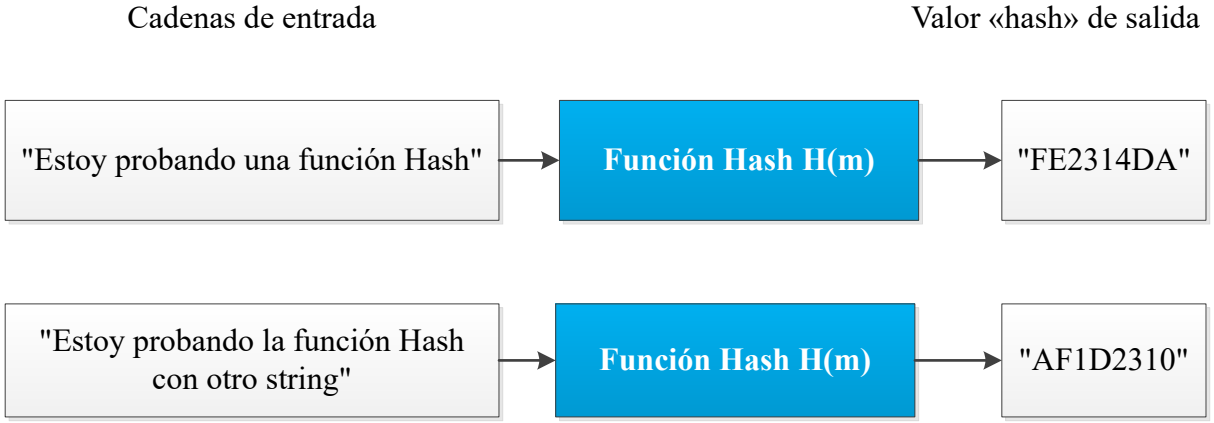

Figura 2.4

Este elemento "hash" es el componente de una firma digital que nos permitirá saber si el documento firmado fue alterado o no.

2.1.2.2. Conseguir autoría determinable y no-repudio de la firma digital

Para lograr que una determinada firma digital solo pueda ser realizada por una persona, y por consiguiente evitar el repudio de la misma, esta firma tiene que estar hecha con un algoritmo que incluya algo que solo esa persona conoce: Una clave, que es privada.

Este algoritmo debe hacer que la firma digital posea un formato que solo se consigue utilizando la clave privada del firmante, y que mediante otro algoritmo complementario podamos verificar con qué clave fue realizada esa firma. Esto se consigue utilizando un método criptográfico.

En consecuencia, una firma digital es producto de una función Hash sobre un documento, mensaje o texto, y una encriptación asimétrica (más adelante en este mismo proyecto se profundizará el tema de encriptación simétrica y asimétrica, y los fundamentos de cada una en firmas digitales). Podemos apreciar el proceso de generación de una firma digital en el siguiente diagrama:

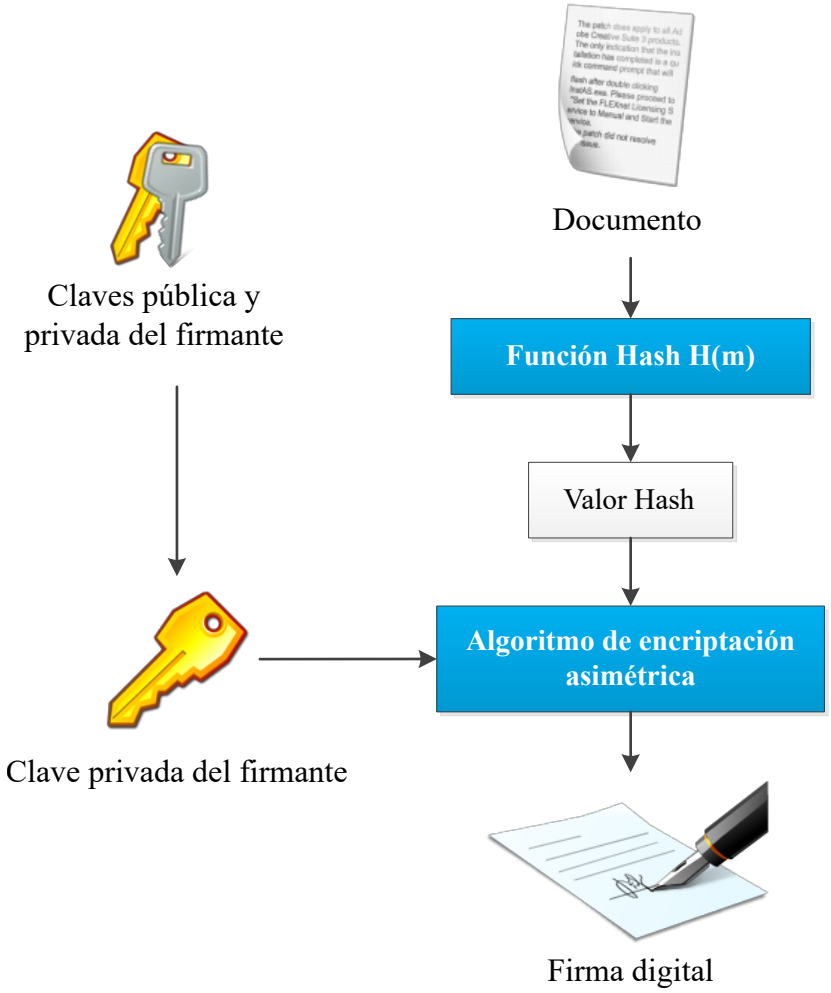

Figura 2.5

Una vez generada la firma digital, el firmante puede distribuir su documento junto con la firma a los demás usuarios, garantizando la autenticidad, integridad y no repudio del mismo. Los demás usuarios podrán realizar una verificación de la firma y el documento mediante un algoritmo de des-encriptación utilizando la clave pública del firmante. Podemos apreciar este procedimiento en el siguiente diagrama de la figura 2.6

Para sintetizar el empleo y funcionalidad de la firma digital, es importante remarcar que esta herramienta informática no garantiza la confidencialidad del mensaje que se transmitirá, ya que este no sufre conversión o encriptación de ningún tipo. La firma digital es un elemento que acompañará al mensaje, mediante el cual el receptor tendrá una garantía de que el mensaje que recibió no sufrió alteraciones de ningún tipo, y que su autor es efectivamente quien afirma serlo.

En el siguiente capítulo se tratarán nociones generales sobre criptografía simétrica y asimétrica, matemática de curvas elípticas, criptografía de curvas elípticas y finalmente, la implemen-

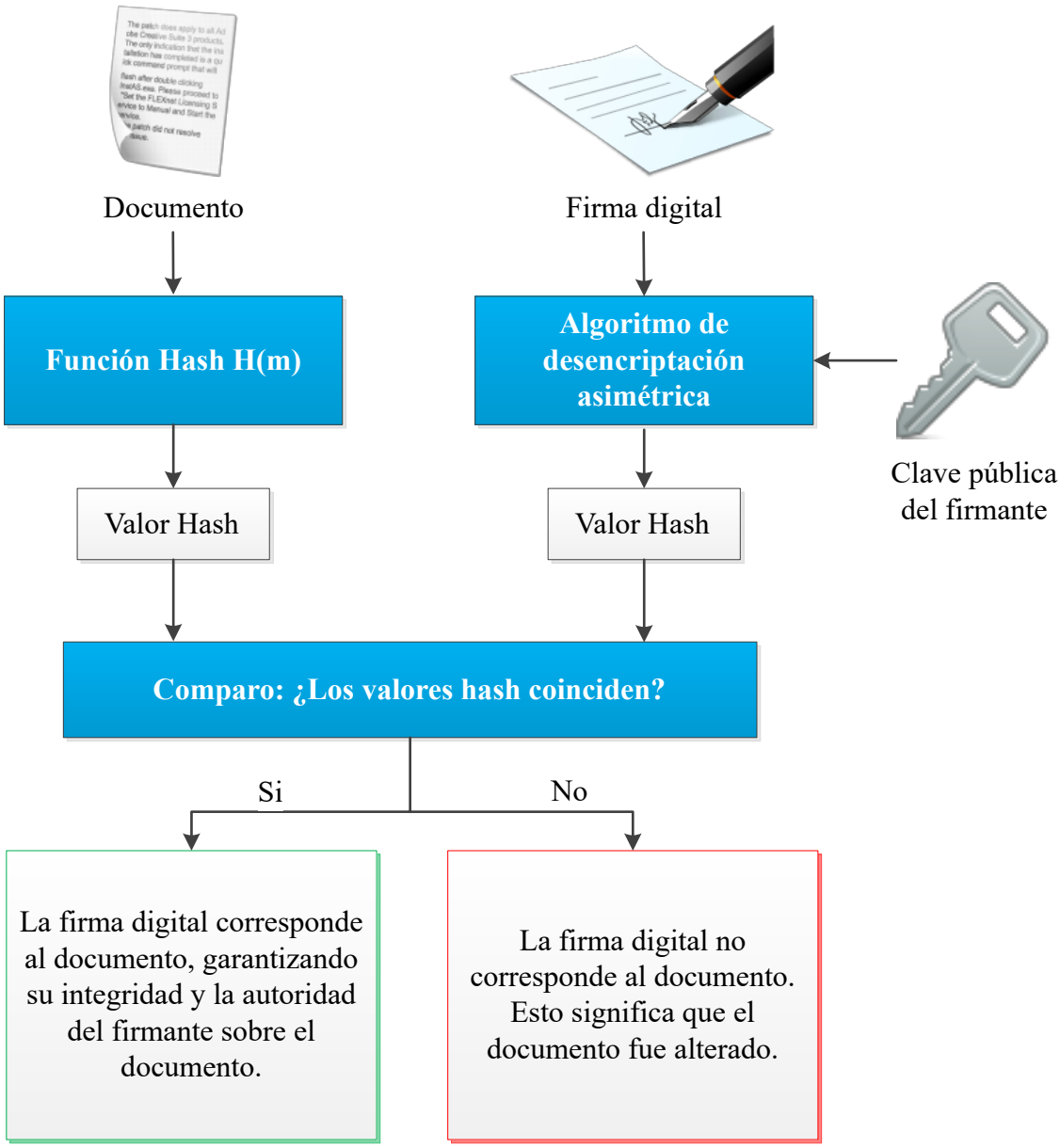

Figura 2.6

tación de la plataforma de generación y verificación de firmas digitales.

2.2. Criptografía

Al hablar de Firma Digital, es importante comprender ciertos conceptos importantes, para fundamentar la elección de un esquema criptográfico que dará soporte al proyecto.

Para comprender la idea básica de criptografía vamos a plantear un ejemplo de comunicación de computadoras mediante una canal imaginario, tal como lo plantean Hankerson, Menezes y Vanstone, en su obra “Guide to Elliptic Curve Cryptography”[2] . Observemos la figura 2.7.

En este diagrama, el usuario A, se comunica con la entidad B, a través de un canal inseguro. Esto significa que la comunicación establecida permite la existencia de una tercera entidad o adversario, llamémosle C, que venza cualquier sistema de seguridad entre A y B, y pueda conocer el contenido de dicha comunicación sin el consentimiento de A o B.

[2]Darrel Hankerson, Alfred J. Menezes y Scott Vanstone. 2010. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Inc. ISBN: 0-387-95273-X.

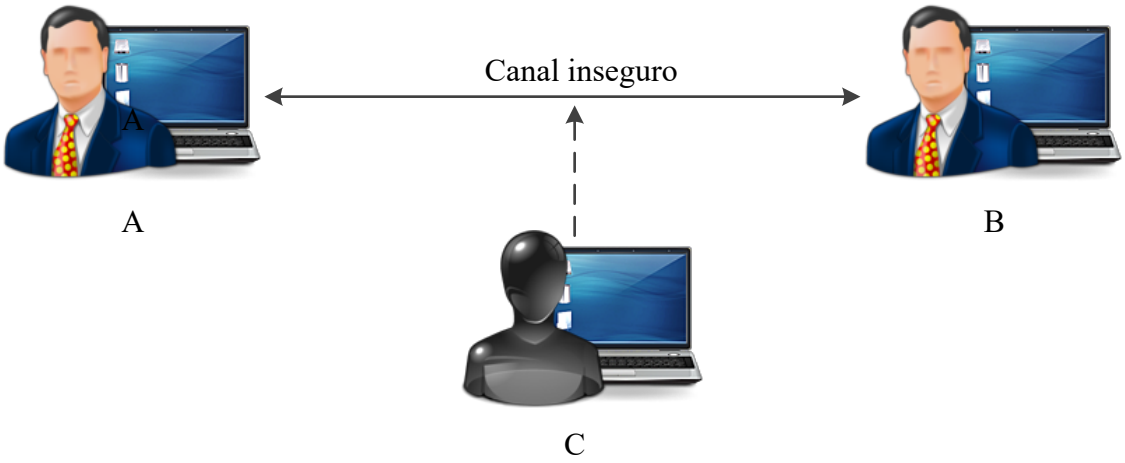

Figura 2.7

Como ejemplo de este caso, se puede mencionar a un individuo “A” que mediante el explorador de internet desea realizar una compra a la empresa “B”, a través de su sitio web. En este escenario, el canal de comunicación es Internet. Un adversario C puede intentar “leer” la información intercambiada entre A y B y así conocer datos sensibles como el número de tarjeta de crédito de “A”, alterar el pedido realizado, o cualquier otra acción que pueda perjudicar tanto a “A” como a “B”. Por esta razón, prácticamente todos los sitios web de la actualidad en los que se realicen transacciones monetarias o de información sensible, adoptan un sistema de seguridad para brindarles la confianza a sus clientes de que su información no será interceptada por terceros.

Los objetivos fundamentales de las comunicaciones seguras son los siguientes:

1. Confidencialidad: Mantener en secreto todos los datos comunicados, excepto para aquellos autorizados a verlos. Esto es, los mensajes enviados de A a B no deberían poder ser interceptados por C.

2. Integridad de datos: Asegurar que los datos en transmisión no serán alterados por C, y que en caso de haber una modificación, B sea capaz de detectar que C realizó modificaciones. Bajo este concepto de integridad de datos, es que se fundamentan las firmas digitales que más adelante en este trabajo trataremos.

3. Autenticación del origen real de los datos: Corroborar el verdadero origen de los datos. B debe ser capaz de verificar que fue A quien le envió los datos y no C.

4. Autenticación de la entidad: Corroborar la identidad del emisor de los datos. B debe ser capaz de validar la identidad de A, sabiendo que no es otra entidad.

5. No repudio: Prevenir a las entidades de negar la realización de una acción. Cuando B recibe un mensaje de A, B puede convencer a una tercera entidad neutral que recibió dicho mensaje. Por consiguiente, A no puede negarle a esta tercera entidad que le envió un mensaje a B.

Dependiendo del área en el que se implemente el sistema de criptografía, se pondrá énfasis en uno o más objetivos de los previamente mencionados. En firmas digitales, utilizamos criptografía principalmente para asegurar la integridad de los datos (objetivo 2), para autenticar la identidad de la entidad (objetivo 4) y con un objetivo de no repudio (objetivo 5).

2.2.1. Modelo de adversario

Los sistemas de criptografía son desarrollados porque necesitamos ocultar cierta información a un tercero. Es por esta razón que debemos modelar a este tercero, llamémosle "adversario", para saber cómo diseñar un sistema de criptografía.

Al momento de realizar nuestro diseño modelamos un adversario realista para que, cuando tengamos que llevar a la práctica nuestro sistema de criptografía, este se encuentre preparado para enfrentar ataques de terceros en la vida real.

Consideramos que nuestro adversario tiene las siguientes características:

- Posee capacidades computacionales considerables: No hay que subestimar las capacidades computacionales del adversario. Los algoritmos de criptografía que diseñamos deben estar pensados para que su "rompimiento" involucre operaciones matemáticas y computacionales de enorme complejidad (este es el ejemplo de la encriptación RSA), y por lo tanto demande capacidades de cálculo equivalentes.
- Puede superar todo tipo de seguridad que exista en el canal de comunicación entre A y B, e interceptar la información que se esté intercambiando.
- Puede modificar la información mientras se esté transmitiendo o enviar sus propios mensajes.
- Tiene conocimientos sobre todos los protocolos y mecanismos criptográficos utilizados entre A y B.

El desafío que propone este modelo impulsa a la creación de mecanismos de seguridad cada vez más poderosos, para tomar ventaja sobre este adversario teórico.

2.2.2. Sistemas criptográficos de clave privada/simétrica y pública/asimétrica

Los sistemas criptográficos, a grandes rasgos, pueden ser clasificados en dos tipos principales: Sistemas criptográficos de clave privada o simétrica, por una parte, y de clave pública o asimétrica, por otra.

2.2.2.1. Sistemas criptográficos de clave privada o simétrica

En los sistemas de clave privada o simétrica, las entidades en comunicación se ponen de acuerdo sobre la clave a utilizar en la comunicación. Esta información es tanto secreta como auténtica. Luego, usan un esquema de encriptación simétrico para conseguir la confidencialidad deseada en los mensajes transmitidos.

En la figura 2.8 podemos observar una comunicación con criptografía de clave simétrica:

Un ejemplo de comunicación utilizando criptografía de clave simétrica consistiría en la siguiente serie de pasos:

1. Previo a cualquier comunicación, A y B acuerdan una clave "k", que comparten a través de un canal autenticado y secreto.
2. A formula un mensaje "m" a transmitir.
3. A codifica el mensaje utilizando, por ejemplo, una función "ENC" y la clave "k". El mensaje cifrado entonces resultaría $c = ENC_k(m)$.
4. El mensaje cifrado "c" es transmitido desde A hacia B a través de un canal inseguro.

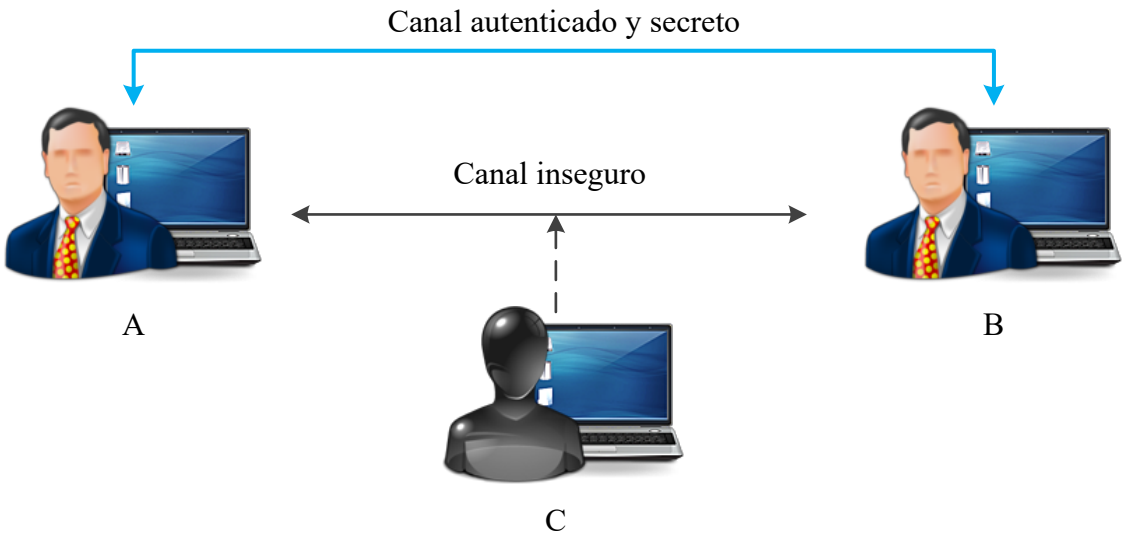

Figura 2.8

5. B recibe el mensaje cifrado “c” y lo decodifica utilizando una unción “DEC” y la misma clave previamente compartida, con la que el mensaje fue cifrado. $m = DEC_k(c)$.

En caso de que C intercepte el mensaje enviado a través del canal inseguro, no lo comprenderá porque no tiene la clave previamente compartida “k”. Esta forzado a utilizar algoritmos para romper la codificación.

2.2.3. Inconvenientes: Distribución y administración de claves

La principal ventaja de los sistemas de clave simétrica es la alta eficiencia. Compone un esquema de comunicación relativamente simple al utilizar una sola clave que tanto A como B utilizan para codificar y decodificar los mensajes.

Pero los sistemas de clave simétrica conllevan tres inconvenientes:

- Distribución de claves: A y B tienen que coordinar una clave antes de comenzar cualquier comunicación, mediante un canal autenticado y secreto. Es decir, un canal conocido y de confianza para A y B, en el que C no pueda infiltrarse para interceptar la clave “k” que se está compartiendo. Si C consigue interceptar la clave “k” compartida, podrá decodificar todos los mensajes transmitidos en el canal inseguro.

  A y B deben contrarrestar este problema de distribución de claves utilizando un canal de confianza en el que puedan intercambiar el material de encriptación.

- Administración de claves: En una red de N nodos, cada nodo debería almacenar la clave de encriptación para comunicarse en forma segura con los otros N-1 nodos. Este problema puede ser resuelto incorporando una entidad de confianza a la red que distribuya las claves entre los nodos según estas son solicitadas, pero no es aplicable en todos los escenarios.

- Inadecuado para firmas digitales: Los sistemas de clave simétrica son inadecuados para ser implementados en firmas digitales, porque se pierde una de las principales características que poseen estas últimas: El no-repudio.

  Un adversario puede firmar un documento utilizando una clave “k” que no le pertenece y simular ante otras entidades, sin posibilidad de repudio, que el documento fue firmado por el verdadero propietario de “k”. Por esta razón, la opción más adecuada para un sistema de firmas digitales es sistema de clave pública, o asimétrica.

2.2.3.1. Sistemas criptográficos de clave pública o asimétrica

La criptografía de clave asimétrica fue introducida en 1975 por Diffie, Hellman y Merkle, con el objetivo de diseñar un sistema criptográfico sin los inconvenientes que tenían los de clave simétrica (mencionados anteriormente).

En los sistemas de clave simétrica, las entidades comunicantes (A y B) intercambian material de encriptación (es decir, claves), que es auténtico pero no secreto. Cada entidad selecciona un par de claves (e, d) que consiste en una clave pública "e", y una clave privada relacionada "d" que mantienen en secreto. Estas claves se caracterizan principalmente porque es muy simple determinar la clave pública a partir de la clave privada, pero es prácticamente imposible de determinar computacionalmente la clave privada solamente conociendo la clave pública.

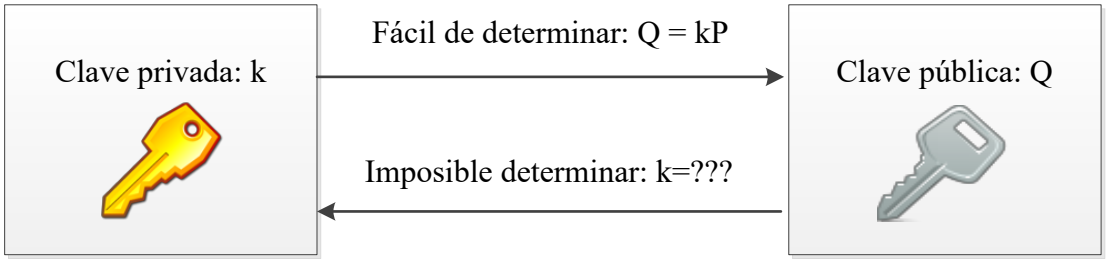

Figura 2.9

El esquema que describe una comunicación utilizando encriptación asimétrica es muy similar al de encriptación simétrica, con la diferencia de que utiliza un canal solamente autenticado (no secreto):

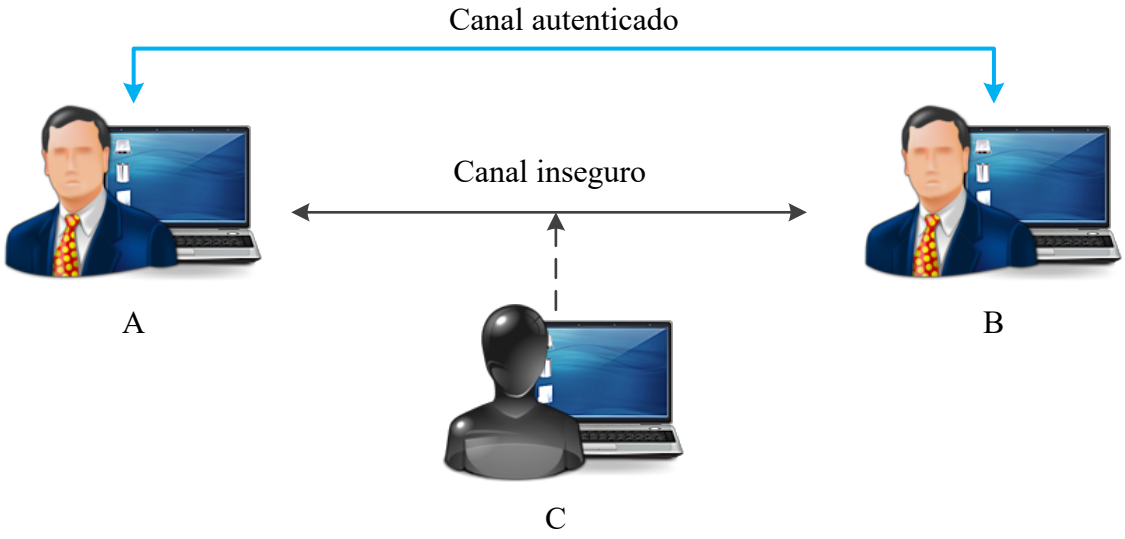

Figura 2.10

Los siguientes pasos ejemplifican una comunicación entre A y B utilizando encriptación asimétrica:

1. A obtiene una copia de la clave pública de b "e_b" a través de un canal autenticado. Esto quiere decir que A tiene que estar seguro que la clave pública transmitida pertenece a B y no a C.

2. A formula un mensaje "m" para transmitirle a B.

3. A genera el mensaje cifrado utilizando una función "ENC" y la clave pública "e_b".$c = ENC_{e_b}(m)$.

4. El mensaje "c" es transmitido de A a B a través de un canal inseguro.

5. B utiliza una función "DEC" y su clave privada "d_b" para desencriptar el mensaje "m". $m = DEC_{d_b}(c)$.

*. Un adversario C que intercepte el mensaje "c" y tenga conocimiento solamente de la clave pública "e_b" no podrá desencriptar el mensaje cifrado. Por esta razón comprendemos que no es necesario que las claves públicas sean secretas y pueden ser transmitidas con libertad a través de un canal. El único requerimiento de este, es que sea autenticado. A tiene que estar seguro que la clave pública que obtuvo pertenece a B y no a C. Si A encripta un mensaje utilizando la clave pública de C pensando que es la de B, entonces C podrá interceptar y decodificar los mensajes transmitidos a través del canal inseguro.

Además de esta mejora por sobre el modelo de clave simétrica, los sistemas de clave asimétrica permiten el "no-repudio" del mensaje encriptado. A puede generar un mensaje "m" y una firma digital "s" mediante una función "SIGN":

$$s = SIGN_{d_a}(m)$$

Una vez que B recibe el mensaje "m" y la firma digital "s", teniendo la clave pública de A "e_a" utiliza un algoritmo de verificación de firmas para confirmar que "s" fue efectivamente generado a partir de "m" y "d_a" (la clave privada de A). Como "d_a" es conocido solamente por A, B se asegura que el mensaje fue efectivamente generado por A.

Por esta razón, el esquema de criptografía asimétrica es el más adecuado para la implementación de un sistema de firmas digitales.

A continuación estudiaremos la teoría de curvas elípticas, y luego la implementación de un sistema de criptografía de clave asimétrica basado en curvas elípticas.

2.3. Esquemas criptográficos actuales

La gran mayoría de las Entidades Certificantes y Sistemas de Firma Digital de la actualidad utilizan Encriptación RSA para la generación de Claves, Certificados y Firmas Digitales. Pero a pesar de ser el sistema de encriptación más ampliamente utilizado, existe otro esquema de criptografía que ofrece diversas ventajas que favorecen a los propósitos de este proyecto: **Criptografía de Curvas Elípticas**. Este esquema nos permitirá el desarrollo de un Sistema de Firma Digital computacionalmente económico y de fácil incorporación.

Si la Criptografía de Curvas Elípticas ofrece ventajas por sobre RSA, ¿Por qué es RSA el sistema de criptografía más ampliamente utilizado en el mercado, si la criptografía de curvas elípticas ofrece mayores ventajas? Existen diversos factores que dieron lugar a esto. En primer lugar podemos mencionar que RSA surgió primero históricamente. El sistema fue publicado por primera vez en 1978, y su documentación respecto de sus algoritmos y claves está disponible desde 1993. En contraparte, la idea de utilizar matemática de curvas elípticas para criptografía surgió en 1985, mientras que los estándares y documentaciones recién estuvieron disponibles a fines de los 90'. Para cuando estos estándares fueron publicados, el sistema RSA ya tenía un dominio considerablemente amplio del mercado.

En segundo lugar, la matemática detrás de RSA es más simple de comprender que la matemática de curvas elípticas. Por consiguiente, la implementación de su código es menos dificultosa.

¿Qué ventajas ofrece la Criptografía de Curvas Elípticas por sobre RSA? Son dos las ventajas globales que presenta: Seguridad y rendimiento

En los sistemas criptográficos, la longitud de la clave es uno de los parámetros primordiales a la hora de definir su funcionamiento. Existen diversas organizaciones académicas y privadas que proveen recomendaciones y fórmulas matemáticas para estimar la longitud de clave mínima

para determinadas aplicaciones de seguridad. De entre las recomendaciones y documentaciones más recientes, podemos mencionar:

- ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information), según su documentación "ANSSI Recommendations (2014)" indica que para aplicaciones consideradas "seguras" en períodos mayores al año 2030, se debe implementar un nivel de seguridad de 128 bits simétricos, que es equivalente a 3072 bits en RSA, o 256 bits en Curva Elíptica.
- ENCRYPT (Organización dedicada al estudio de criptografía), en su documentación "ENCRYPT II Recommendations (2012)", establece que para aplicaciones seguras de 2015 a 2040, se debe implementar un nivel de seguridad de 128 bits simétricos, que equivale a 3248 bits de RSA o 256 bits de Curva Elíptica.
- NIST (Agencia de administración de tecnología del Departamento de Comercio de los Estados Unidos) recomienda, para períodos posteriores al año 2030, implementar un nivel de seguridad de 128 bits simétricos, que equivale a 3072 bits en RSA o 256 bits en Curva Elíptica.
- BSI (Bundesamt für Sicherheit in der Informationstechnik) recomienda, para períodos posteriores entre el año 2017 y el año 2021, implementar un nivel de seguridad de 128 bits simétricos, que equivale a 3072 bits en RSA o 256 bits en Curva Elíptica.

De estas recomendaciones podemos observar que las longitudes de clave en Curva Elíptica son considerablemente inferiores en comparación a su equivalente en RSA. Por otra parte, los algoritmos de Curva Elíptica son computacionalmente más "económicos" (menor consumo de CPU y memoria).

2.3.1. Curvas elípticas

2.3.1.1. Introducción

Para brindar una adecuada y comprensiva descripción sobre el concepto matemático de "Curva Elíptica" y cuál es su uso en criptografía, en esta sección sintetizaremos las obras de Llorenç Huguet Rotger, Josep Rifà Coma y Juan Gabriel Tena Ayuso [3]; y Darrel Hankerson, Alfred, Menezes Scott Vanstone [4].

El estudio de curvas elípticas proviene de mediados del siglo XIX, y hoy en día existe una amplia literatura que describe las propiedades de estos interesantes elementos algebraicos. Hendrik Lenstra describió un ingenioso algoritmo para factorear números enteros, que se basa en las propiedades de las curvas elípticas. Este descubrimiento desencadenó la investigación de otras aplicaciones de curvas elípticas: En teoría de números computacionales y en criptografía.

La criptografía de curva elíptica es una de las disciplinas más prometedoras en el campo de los cifrados asimétricos. Las primeras propuestas de uso de las curvas elípticas en Criptografía fueron hechas por Neal Koblitz y Victor Miller en 1985. Precisamente el principal argumento que se esgrimen los detractores de estas técnicas es que, si bien las curvas elípticas han sido objeto de estudio y análisis durante más de un siglo, las propiedades que pueden estar directamente relacionadas con su calidad como base de un sistema criptográfico, apenas llevan quince años siendo consideradas.

[3] Llorenç Huguet Rotger, Josep Rifà Coma y Juan Gabriel Tena Ayuso. *Criptografía con curvas elípticas*, capítulo 3

[4] Darrel Hankerson, Alfred J. Menezes y Scott Vanstone. 2010. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Inc. ISBN: 0-387-95273-X, prefacio y capítulo 3

2.3.1.2. Concepto de curva elíptica

Una curva elíptica sobre $\mathbb{R}$ es un conjunto de puntos en el plano (x,y) que satisfacen una ecuación de la forma:

$$y^2 = x^3 + ax + b$$

Donde “x”, “y”, “a” y “b” son números reales. Esta ecuación es llamada también “ecuación de Weierstrass”. Cada elección de valores de “a” y “B” configura una curva elíptica diferente, es decir, identifican unívocamente cada curva. En las figuras 2.11 y 2.12 podemos observar dos ejemplos de curvas elípticas

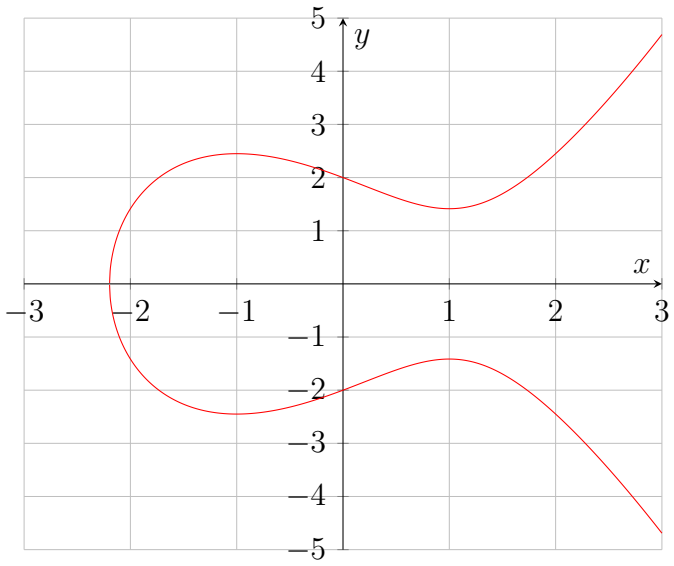

Figura 2.11: $y^2 = x^3 - 3x + 4$

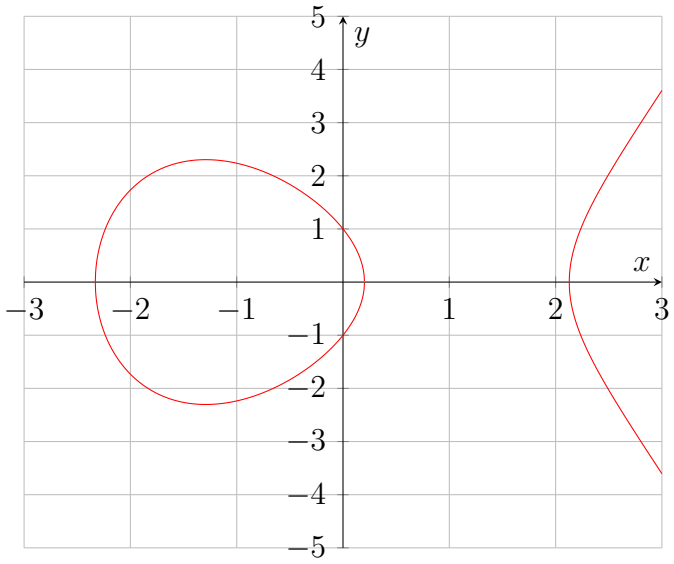

Figura 2.12: $y^2 = x^3 - 5x + 1$

Si $x^3 + ax + b$ no tiene raíces múltiples, lo que significa que $4a^3 + 27b^2 \neq 0$, entonces la curva correspondiente, en conjunción con un punto especial O, llamado punto en el infinito, más la operación de suma que trataremos mas adelante, configuran lo que vamos a denominar

"grupo de curva elíptica $E(R)$". Hay que resaltar que O es un punto situado por encima del eje de las abscisas a una distancia infinita, y que por lo tanto no tiene valor concreto.

2.3.1.3. Operación Suma en $E(\mathbb{R})$

Definir una operación de suma para el conjunto sobre el que queremos trabajar es el primer paso para definir la composición interna del conjunto propiamente dicho. La operación suma está dada a partir de que, dados dos elementos cualesquiera, nos devuelva otro elemento que también pertenezca al conjunto. Denominaremos a esta operación como "suma", y la representaremos mediante el signo "+", en forma similar a la suma algebraica que realizamos comúnmente sobre números reales.

Sean dos puntos $P = (P_x, P_y)$ y $Q = (Q_x, Q_y)$, la operación suma se define de la siguiente manera:

- $P + 0 = 0 + P = P$, para cualquier valor de P. Esto significa que 0 se comporta como el elemento neutro de la operación suma.
- Si $P_x = Q_x$ y $P_y = -Q_y$, decimos que P es el opuesto de Q, lo que se representa como P=-Q, y además P+Q=Q+P=O, por definición.
- Si $P \neq Q$ y $P \neq -Q$, entonces la suma de estos puntos está dada por la traza de una línea recta que une P con Q. Esta recta cortará la curva en un punto. El opuesto de este punto, es el punto R, resultado de la suma P+Q=R. En la figura 2.13 podemos apreciar una suma de dos puntos sobre una curva elíptica.

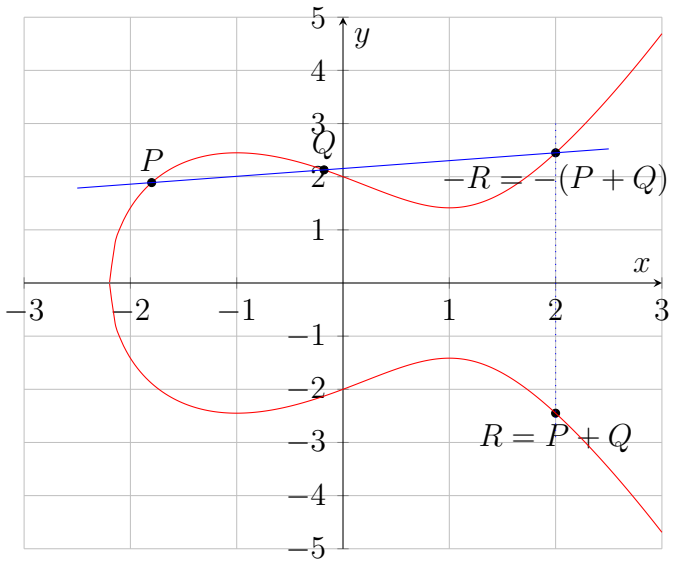

Figura 2.13

- Para sumar un punto P consigo mismo, se utiliza la tangente a la curva en P. Si $P_y \neq 0$, dicha tangente cortará a la curva en un único punto. La suma R=P+P será el opuesto de dicho punto. Podemos apreciar la duplicación o suma de un punto sobre si mismo en una curva elíptica, en la figura 2.14.
- Para una suma de un punto P consigo mismo cuando $P_y = 0$, la tangente a la curva será vertical, perpendicular al eje "y". Entonces consideramos que la tangente corta a la curva en el infinito. Por lo tanto P+P=O si $P_y = 0$.

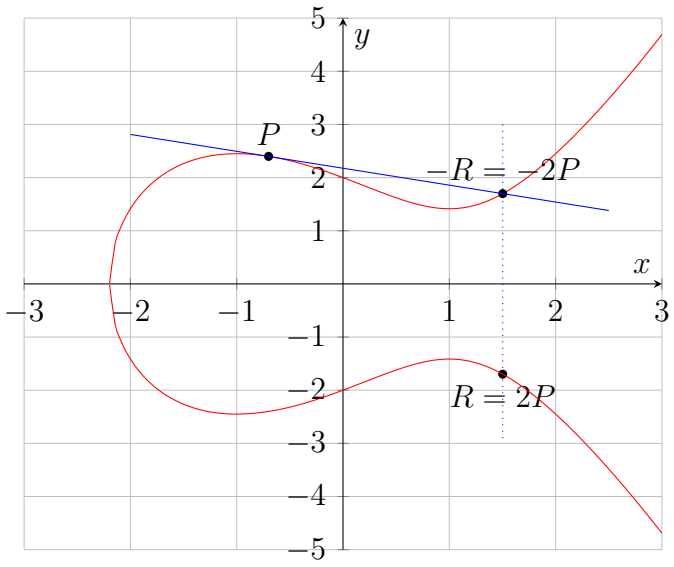

Figura 2.14

De lo anterior podemos notar dos cosas:

- Determinamos que sumar a un punto P consigo mismo k veces, es equivalente a multiplicar a P por el escalar k, y lo notaremos por kP.
- Cuando se suma P y $-P$, la recta que une a ambos puntos resulta perpendicular al eje de las abscisas, por lo que cortará a la curva en el punto infinito, dando como resultado O.

Algebraicamente, la suma de curvas elípticas se define mediante las siguientes ecuaciones:
Sean P=(P_x,P_y) y Q=(Q_x,Q_y) dos puntos ubicados en una determinada curva elíptica, la suma P+Q está dada por:

$$P + Q = (R_x, R_y)$$
$$R_x = s^2 - P_x - Q_x$$
$$R_y = -P_y + s(P_x - R_x)$$

Donde:

$$s = \frac{P_y - Q_y}{P_x - Q_x}$$

La operación de suma de P sobre sí mismo, o Doble de P está dada por:

$$2P = (R_x, R_y)$$
$$R_x = s^2 - 2P_x$$
$$R_y = -P_y + s(P_x - R_x)$$

Donde:

$$s = \frac{(3P_x^2 + a)}{2P_y}$$

Podemos observar que el valor “s” representa a la pendiente de la recta que une P y Q, o bien la tangente del punto P. Estas operaciones resultan en puntos con coordenadas en $\mathbb{R}$, que no son de nuestro interés para criptografía por una cuestión de precisión de cálculo de valores

decimales y márgenes de error. Necesitamos resultados expresados en coordenadas enteras, que son valores útiles para nuestros propósitos en criptografía. Para ello, introduciremos las curvas elípticas en cuerpos de Galois (F_p).

2.3.1.4. Curvas elípticas en F_p

Un cuerpo de Galois F_p es un cuerpo finito generado por p, siendo p un número primo. En este conjunto todos los elementos, menos el cero, tienen un elemento inverso. Por esta razón podemos realizar operaciones de suma, resta, multiplicación y división exactamente de la misma manera que en los números reales. Podemos calcular qué puntos satisfacen una curva elíptica en un cuerpo de Galois F_p mediante la ecuación:

$$y^2 = x^3 + ax + b\ (mod\ p)$$

De esta forma, definimos el conjunto $E(F_p)$. Para comprender mejor, podemos ejemplificar mediante al curva elíptica $y^2 = x^3 + x\,(mod 23)$. Esta curva elíptica está dada por $a = 1$ y $b = 0$.

Los 23 puntos que satisfacen esta ecuación son:

(0,0)	(1,5)	(1,18)	(9,5)	(9,18)
(11,10)	(11,13)	(13,5)	(13,18)	(15,3)
(15,20)	(16,8)	(16,15)	(17,10)	(17,13)
(18,10)	(18,13)	(19,1)	(19,22)	(20,4)
	(20,19)	(21,6)	(21,17)	

Gráficamente, estos puntos los podemos apreciar de la siguiente manera:

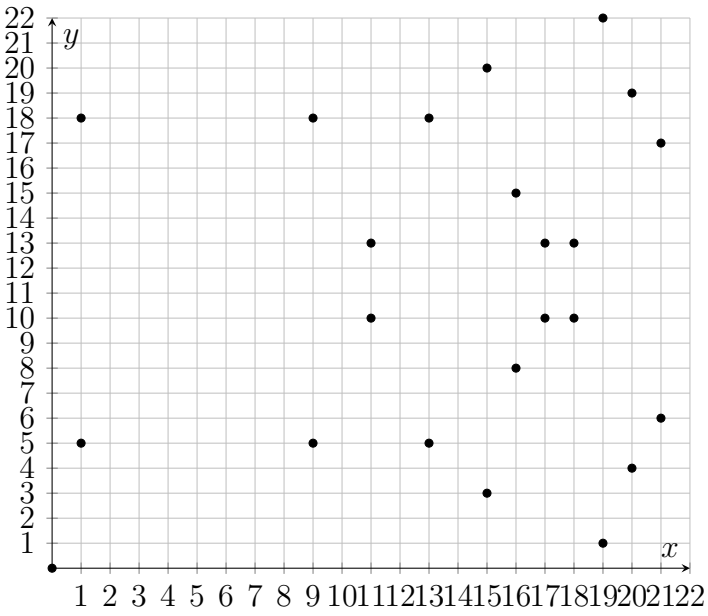

Figura 2.15

Podemos observar que dado un punto de la curva (x, y), el valor $(x, -y)\ (mod\ n)$ también pertenece a esta. De esta manera, hay dos puntos para cada valor de "x". En el campo F_p, la operación de suma está dada por:

$$P + Q = (R_x, R_y)$$
$$R_x = s^2 - P_x - Q_x\ (mod\ p)$$
$$R_y = -P_y + s(P_x - R_x)\ (mod\ p)$$

Donde:

$$s = \frac{P_y - Q_y}{P_x - Q_x} \ (mod\ p)$$

Y la operación de suma de P sobre sí mismo, o Doble de P, sobre el campo F_p está dada por:

$$2P = (R_x, R_y)$$
$$R_x = s^2 - 2P_x \ (mod\ p)$$
$$R_y = -P_y + s(P_x - R_x) \ (mod\ p)$$

Donde:

$$s = \frac{3P_x^2 + a}{2P_y} \ (mod\ p)$$

2.3.1.5. El problema del Logaritmo Discreto en Curvas Elípticas (ECDLP)

Si tomamos un punto P cualquiera de una curva elíptica, denominaremos $\{P\}$ al conjunto $\{O, P, 2P, 3P, \ldots\}$. En grupo F_p los conjuntos de esta naturaleza deberán ser finitos, ya que el número de puntos de la curva es finito. Por lo tanto si disponemos de un punto $Q \in \{P\}$, debe existir un entero k tal que $Q = kP$. El Problema del Logaritmo Discreto en Curvas Elípticas (Elliptic Curve Discrete Logarythm Problem) consiste en la dificultad de hallar el número k conociendo solamente P y Q. Hasta el momento, no existe ningún algoritmo eficiente para calcular el valor de k conociendo solamente P y Q y la curva a la que pertenecen. Este problema es la base que utilizaremos para el desarrollo de algoritmos criptográficos de clave pública.

Algoritmos e implementación de operaciones aritméticas sobre curvas elípticas En este capítulo demostraremos los algoritmos utilizados para la suma, duplicación y multiplicación de puntos de una curva elíptica sobre el campo F_p. Como mencioné anteriormente, el lenguaje elegido es Java, debido a su potencia para la realización de cálculos complejos y manejo de números grandes, además de su versatilidad para el diseño de aplicaciones para múltiples plataformas.

Suma de puntos sobre F_p En el algoritmo de suma tenemos como entradas los puntos sumandos P y Q, y la curva C en la que se encuentran. En primer lugar determinamos si P_x y Q_x son iguales. De ser así, la suma resultaría en un valor en el infinito. Luego calculamos el valor de la tangente "s", y finalmente las coordenadas del punto R resultante, R_x y R_y.

Doble de un punto sobre F_p Para la función utilizada para "doblar" un punto, tenemos como entrada un punto P a doblar, y la curva C donde está definido. En primer lugar se determina si $P_y = 0$, lo que significaría que el doble de este punto se encuentra en el infinito. Si no lo es, entonces procedemos a calcular el valor de la tangente "s", y luego las coordenadas del resultado R_x y R_y.

Multiplicación de un punto sobre F_p La multiplicación de un punto por un escalar sobre F_p se consigue realizando sucesivas sumas y duplicaciones de puntos del campo F_p. Existen diversos algoritmos que buscan atajos para encontrar resultados de sumas, tal como descomponer el escalar en componentes a partir de los cuales realizar sumas iterativas de puntos, o guardar en memoria una serie de puntos precomputados a partir de los cuales realizar operaciones para

conseguir el resultado final. Cada uno de estos algoritmos fue diseñado para funcionar de manera óptima bajo diferentes implementaciones, habiendo algoritmos óptimos para ser aplicados por hardware o software

A la hora de desarrollar un sistema de Criptografía de Curva Elíptica, es de vital importancia elegir los algoritmos que mejor se desenvuelvan bajo el entorno en el que sean implementados.

2.3.2. ECDLP y criptografía de clave asimétrica

Como mencionábamos previamente, la base de la criptografía de clave asimétrica se encuentra en la dificultad de determinar la clave privada solamente conociendo la clave pública, en contraparte con la facilidad de determinar la clave pública conociendo la clave privada.

Para comprender de forma adecuada la implementación criptográfica del concepto de Curvas Elípticas, nos referiremos a los contenidos publicados por Hankerson, Menezes y Vanstone en su libro "Guide to Elliptic Curve Cryptography"[5].

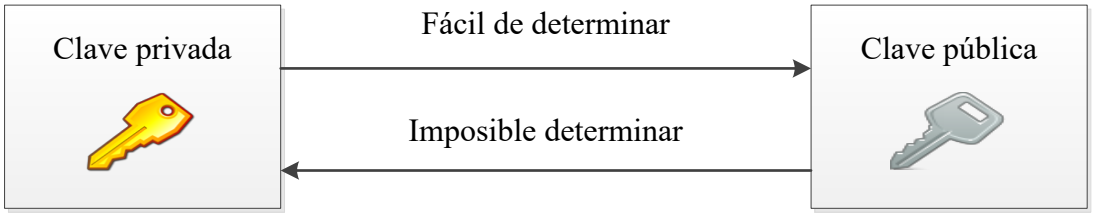

Figura 2.16

Un sistema de criptografía de curvas elípticas utiliza un conjunto de Parámetros Dominio, conformado por una curva elíptica y un punto base P, mediante el cual se determinan las claves públicas y privadas. Es a través de estas claves, que se realizan las encriptaciones sobre la información que sea necesario.

Sabiendo que en el ECDLP, teniendo una multiplicación de puntos $Q = kP$, Q es un valor fácil de determinar conociendo k, y k es un valor extremadamente difícil de determinar solamente conociendo Q y P, podemos establecer una analogía con la criptografía de clave asimétrica en donde k es nuestra clave privada y Q nuestra clave pública. Podemos apreciar esto en la figura 2.17.

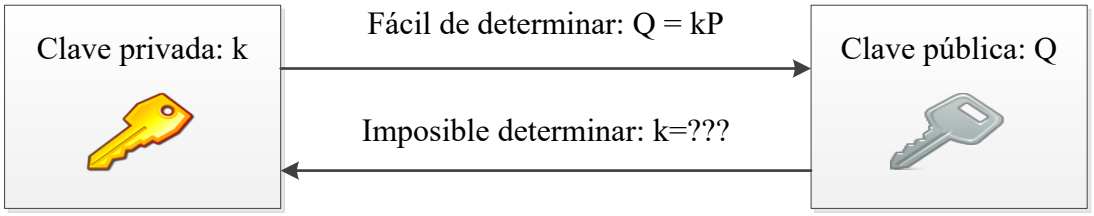

Figura 2.17

Un sistema de criptografía de curvas elípticas se basa en la utilización de una curva elíptica C y un punto base P, que se encuentra en C. Cada usuario elige una clave privada que es un escalar "k", a partir del cual se determina su clave pública calculando $Q = kP$.

2.3.2.1. Parámetros dominio

Los Parámetros Dominio sobre un esquema de curva elíptica describe una curva C definida sobre un campo primo finito F_p, un punto base P que se encuentre en la curva $C(F_p)$ y el orden

[5]Darrel Hankerson, Alfred J. Menezes y Scott Vanstone. 2010. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Inc. ISBN: 0-387-95273-X.

n de dicho punto.

Los parámetros deben ser elegidos de tal manera que nuestro Problema de Logaritmo Discreto sobre Curvas Elípticas sea resistente a cualquier tipo de ataque conocido.

Estos parámetros dominio son $D = (q, a, b, P, n)$, donde:

q: Orden del campo de la curva elíptica.

a: Parámetro "a" de la curva elíptica.

b: Parámetro "B" de la curva elíptica.

G: Punto de base utilizado para nuestro. Es el punto a partir del cual realizamos los cálculos de claves públicas.

n: Orden del punto base P (mínimo valor de k para el cual kP = Infinito). Las claves privadas deben ser elegidas en un rango [1 ; n-1].

2.3.2.2. Cálculo de pares de claves

Como mencionamos previamente, la clave pública es fácilmente calculable mediante una multiplicación del punto base P por la clave privada. El algoritmo de cálculo de claves es el siguiente:

Algoritmo 2.1: Generación de par de claves de Curva Elíptica

Entrada: Parámetros dominio $D = (q, a, b, P, n)$
Salida: Clave pública Q, clave privada d
1 Elegir una clave privada $d \in \mathbb{Z}\ [1, n-1]$;
2 Calcular $Q = dP$;
3 Devolver (Q, d);

Estas claves pública y privada son las que utilizaremos para las tareas de encriptación y des-encriptación.

2.3.2.3. ECDSA: Algoritmo de Firma Digital por Curvas Elípticas (Elliptic Curve Digital Signature Algorithm)

El Algoritmo de Firma Digital por Curvas Elípticas (ECDSA) es la versión utilizando curvas elípticas del algoritmo DSA (Digital Signature Algorithm). Es el esquema de firma digital más ampliamente estandarizado, figurando en los estándares ANSI X9.62, FIPS 186-2, IEEE 1363-2000 e ISO/IEC 15946-2, además de otros estándares menores. La generación de firmas digitales mediante curva elíptica se basa en la utilización de una función hash denotada por H(), y cálculos de encriptación utilizando las propiedades de las curvas elípticas. El algoritmo de ECDSA de

generación de firma digital es el siguiente:

Algoritmo 2.2: ECDSA: Elliptic Curve Digital Signature Algorithm

Entrada: Parámetros dominio $D = (q, a, b, P, n)$, clave privada d, mensaje m
Salida: Firma digital (r, s)

1 Elegir un escalar $k \in \mathbb{Z}$ aleatorio, $[1, n-1]$;
2 Calcular $kP = (x_1, y_1)$ y almacenar el valor x_1 en una variable $X1$;
3 Calcular $r = X1 \ (mod\ n)$. Si $r = 0$ entonces hay un error en los cálculos y hay que volver a 1;
4 Calcular $e = H(m)$;
5 Calcular $s = k^(-1) \ (e + dr) \ (mod\ n)$. Si $s = 0$ entonces hubo un error y hay que volver a 1.;
6 Devolver (r, s);

El algoritmo complementario, para la verificación de la firma digital y comparación con un texto determinado es el siguiente:

Algoritmo 2.3: Verificación de una firma electrónica generada mediante ECDSA

Entrada: Parámetros dominio $D = (q, a, b, P, n)$, clave pública Q, mensaje m, firma digital (r, s)
Salida: Aceptación o rechazo de la firma digital

1 Verificar que r y s son enteros en el intervalo $[1, n-1]$. Si estas verificaciones fallan, devolver (”Se rechaza la firma digital”);
2 Calcular $e = H(m)$;
3 Calcular $w = s^(-1) \ (mod\ n)$;
4 Calcular $u_1 = ew \ (mod\ n)$ y $u_2 = rw \ (mod\ n)$;
5 Calcular $X = u_1P + u_2Q$;
6 **si** $X = Infinito$ **entonces**
7 | Devolver (”Se rechaza la firma digital”)
8 **fin**
9 ;
10 Almacenar en $X1$ la coordenada x de X;
11 Calcular $v = X1 \ (mod\ n)$;
12 **si** $v = r$ **entonces**
13 | Devolver (”La firma digital es correcta”);
14 **fin**
15 **sinó**
16 | Devolver (”Se rechaza la firma digital”);
17 **fin**

2.3.2.4. Comprobación de funcionamiento de la verificación de firmas digitales

El fundamento matemático que nos demuestra que este sistema de verificación funciona es el siguiente: Si una firma digital (r, s) en un mensaje m de verdad fue generada por el firmante legítimo, entonces “s” equivale a $k^(-1) \ (e + dr) \ (mod\ n)$. Acomodando los valores, tenemos:

$$k = s^{-1}(e + dr) = s^{-1}e + s^{-1}rd = we + wrd = u_1 + u_2d \ (mod\ n)$$

Entonces, $X = u_1P + u_2Q = (u_1 + u_2d)P = kP$, y por lo tanto v=r, como es requerido para que la comparación de la firma digital con el mensaje m sea correcta.

2.4. PKI, clases de infraestructuras y nuevas propuestas

El despliegue de sistemas criptográficos en redes de computadoras modernas se utilizó históricamente para solventar dos problemáticas fundamentales. En primer lugar, la autenticación de usuarios en dichas redes. Los usuarios utilizan contraseñas, dispositivos de "One-Time Password" y biométricos para autenticarse en computadoras o sistemas de una determinada red. La encriptación es utilizada tanto para proteger la transmisión de datos, como para métodos de autenticación mediante credenciales.

En segundo lugar, los sistemas criptográficos son empleados para el despliegue de Infraestructuras de Clave Pública (PKI). Esta tecnología asume que cada cliente debería poseer un par de claves asimétrico. La clave pública puede ser compartida con otras contrapartes, la cual puede autentificar la pertenencia de la clave privada, verificar la firma del dueño realizada sobre un bloque de datos, o encriptar datos que solo el dueño del par de claves puede desencriptar. Un par de claves, entonces, brinda seguridad "uno a muchos", en vez de la forma clásica de "uno a uno", obtenida utilizando criptografía de clave simétrica.

Una estructura básica de infraestructura criptográfica puede detallarse como una conjunto de nodos enlazados por canales de comunicación, donde un nodo puede ser una computadora o una persona, y el canal de comunicación puede ser una red local. Cada nodo es poseedor y manipula sus propias claves para la realización de comunicaciones seguras dentro de la infraestructura. La problemática que esto conlleva es que la seguridad de esta infraestructura depende, en gran parte, de las medidas de seguridad que adopten los nodos con respecto al almacenamiento de sus claves privadas. Por esta razón, se propone una alternativa al diseño de la clásica Infraestructura de Clave Pública: Infraestructura de Servidores Delegados.

2.4.1. Servidores delegados

En el año 2002, Trevor Perrin, Logan Bruns, Jahan Moreh y Terri Olkin publicaron un reporte investigativo detallando una infraestructura de clave pública (PKI) basada en delegación de operaciones criptográficas en un "servidor delegado"[6].

Mediante esta infraestructura, se propone destinar un servidor para el almacenamiento y manipulación de las Claves Privadas de los usuarios, al cual ellos delegan todas las tareas criptográficas. Los usuarios solo dispondrán de credenciales de acceso al servidor, en el cual se realizarán las tareas de, por ejemplo, firmado, verificación, encriptación y desencriptación.

Este esquema ofrece la ventaja de centralizar la seguridad del sistema en un solo servidor, y desligar a los usuarios de la responsabilidad de asegurar sus claves privadas. La problemática que este esquema conlleva es que el correcto diseño del servidor se vuelve crucial para garantizar la seguridad y eficiencia de todo el resto de la estructura.

[6]Trevor Perrin et al. 2002. "Delegated cryptography, online trusted third parties, and PKI". En: *1nd Annual PKI Research Workshop*. Citeseer.

Capítulo 3

Definición del problema

En una organización o equipo de trabajo donde una de las actividades más significativas durante el desarrollo del negocio, es el intercambio de documentos digitales a través de diferentes sistemas informáticos de comunicaciones, es crucial que esto se realice de manera segura. La Firma Electrónica es un instrumento útil a la hora de garantizar la autenticidad e integridad de los documentos digitales transmitidos.

Si esta misma organización desea implementar el uso de Firma Electrónica, todavía no poseyendo una Infraestructura de Clave Pública (comúnmente denominada "PKI", por sus iniciales en inglés), deberá efectuar un esfuerzo importante para conseguirlo. En primer lugar, deberá asignar ciertas áreas de la organización como Autoridades de Certificación, y Autoridades de Registro. Luego, deberá definir protocolos u operatorias para que los miembros de la organización soliciten la creación de sus claves públicas y privadas y la creación de sus certificados. Muy probablemente existan miembros de la organización que no estén entrenados para la correcta manipulación de sus claves pública y, más gravemente, privada. Consecuentemente, se deberá invertir un esfuerzo grande en capacitarlos. Finalmente, debido a que la organización utiliza diversos sistemas de comunicaciones, se deberá consensuar la utilización de software específicos para el firmado de diferentes tipos de archivo. Cada miembro de la organización deberá recurrir a cada software especificado para realizar la generación y verificación de firmas electrónicas.

Permitir a usuarios con poco entrenamiento manipular sus claves privadas puede conllevar un grave riesgo, principalmente en lugares de trabajo donde otros miembros de la organización pueden llegar a tener acceso a computadoras ajenas. La utilización de SmartCards o Tokens USB para manipular claves privadas es una de las mejores alternativas en estos casos, pero puede resultar económicamente costoso y tedioso de implementar en ciertas organizaciones.

Trataremos de encontrar una solución que facilite la implementación en una Infraestructura de Clave Pública de una organización, y que permita a los miembros efectuar generación y validación de firmas electrónicas en una única plataforma de manera fácil e intuitiva, y evitando que los mismos manipulen directamente sus claves privadas y minimizar las vulnerabilidades de seguridad que ésto puede acarrear.

3.1. Objetivo

El objetivo del presente proyecto consiste en planificar, diseñar y desarrollar un sistema informático que permita a los usuarios generar y verificar firmas electrónicas, además de poder gestionar y almacenar los pares de claves utilizados para efectuar las tareas de firmado, de manera centralizada. Este sistema informático podrá ser desplegado en cualquier organización que desee implementar el uso de firmas electrónicas para garantizar la autenticidad y autoría de los documentos transferidos de forma interna.

El sistema informático a desarrollar, al realizar tareas criptográficas de manera centralizada, almacenará claves públicas y privadas de sus usuarios. Es por esta razón que los aspectos de seguridad del diseño serán centrales para el presente proyecto de grado.

Para conseguir el desarrollo del sistema informático, se detallará completamente su análisis, diseño, desarrollo e implementación, justificando debidamente la elección de las alternativas tecnológicas y metodológicas inherentes al desarrollo de un sistema informático y de un esquema criptográfico.

Como esquema criptográfico a implementar en el presente proyecto, se propondrá la utilización de Criptografía de Curvas Elípticas, analizando sus cualidades y ventajas frente a otros esquemas criptográficos.

3.2. Alcance

Como principal eje de trabajo del presente proyecto de grado, se diseñará un Sistema Interno de Firma Electrónica de carácter centralizado, que almacene de forma segura las Claves Privadas de todos los usuarios involucrados, y que les permita realizar todas sus tareas criptográficas en un ambiente centralizado. Para conseguir efectivamente la conclusión de este sistema, se efectuarán tareas de planificación, análisis, diseño y desarrollo de un prototipo que conformará el Sistema Interno de Firma Electrónica.

Serán elegidos, proporcionando fundamentos, una metodología que permita ejecutar el proyecto de una manera rápida y dinámica, herramientas de análisis y modelado que permitan describir de manera adecuada el diseño realizado, y tecnologías que ayuden a conseguir un sistema funcional.

El sistema a desarrollar será, a su vez, de carácter interno. Este será diseñado para ser implementado y utilizado en el ámbito de una organización que necesite brindar seguridad a sus comunicaciones digitales, más particularmente al intercambio de archivos.

Una vez desarrollado el prototipo del Sistema Interno de Firma Electrónica, se realizará un análisis sobre diversos aspectos de seguridad involucrados en el mismo, y se concluirá con una verificación experimental, donde crearemos un par de claves mediante un usuario del sistema, le asignaremos un certificado firmado por una Autoridad Certificante, y generaremos una firma electrónica de un archivo.

Capítulo 4

Solución Propuesta

La solución que se propone, a partir de la problemática analizada, es la implementación de un Sistema de Firma Electrónica Interno que permita a los usuarios efectuar operaciones de firmado electrónico y verificación de archivos en una plataforma web, accesible desde cualquier explorador web. Empleando esta plataforma, los usuarios podrán generar firmas electrónicas "detached" (esto es, firmas electrónicas sueltas, en archivos separados) de los archivos que requieran. De esta forma, cuando se requiera garantizar la autoría e integridad de un determinado archivo al ser transmitido mediante un medio digital de comunicación, el emisor enviará el archivo junto con la firma electrónica generada.

El Sistema de Firma Electrónica a implementar poseerá las siguientes características:

Sistema que emplea criptografía de Curvas Elípticas Las operaciones criptográficas del Sistema de Firma Electrónica serán implementadas utilizando el esquema criptográfico de Curvas Elípticas, detallado en sección 2.3.2. Los fundamentos matemáticos y criptográficos que dan soporte a este esquema de criptografía serán desarrollados en su totalidad en una "Librería Criptográfica".

Criptografía delegada En la solución que se propone, los usuarios no tendrán posesión directa de sus claves pública y privada para la generación y verificación de firmas electrónicas. Implementando los conceptos propuestos por Perrin, Bruns, Moreh y Olkin (2002), el sistema conformará una entidad de confianza mantenida por la organización, a la cual los usuarios delegarán las operaciones de criptografía asimétrica. De esta manera, el sistema almacenará de forma segura las claves privadas de los usuarios, y realizará las operaciones criptográficas que estos requieran. Los usuarios realizarán el firmado y verificación de archivos a través de la plataforma web, en el servidor.

Ingreso a la plataforma web con verificación de dos pasos Los usuarios ingresarán a la plataforma web utilizando verificación de dos pasos. Emplearán un nombre de usuario, una contraseña y una clave basada en el tiempo, generada mediante una aplicación desde el dispositivo móvil del usuario. De esta manera, la seguridad del acceso al sistema estará basada en un factor de conocimiento (la contraseña) y un factor de posesión (el dispositivo móvil configurado para generar las claves basadas en el tiempo).

Sistema desarrollado mediante arquitectura "Cliente - Servidor" El sistema será desarrollado en un modelo de arquitectura "Cliente - Servidor". El servidor proveerá los métodos necesarios para realizar todas las operaciones de un Sistema de Firma Electrónica y dará persistencia a toda la información necesaria para su funcionamiento, a través de una base de datos. La aplicación "cliente" será una plataforma web que proveerá las interfaces para facilitar su uso al

usuario. Toda comunicación entre la aplicación "cliente" y el servidor se realizarán empleando el protocolo ***https***, de manera que se realice de forma secreta y segura.

4.1. Metodologías a emplear

Diversas características del presente proyecto de grado hicieron que se opte por emplear una metodología de desarrollo ágil para conseguir realizar el mismo. En primer lugar, el hecho de definir objetivos, alcance, problemáticas y soluciones propuestas, previo al comienzo de este proyecto, se asemeja mucho a las "entrevistas" que un desarrollador puede llegar a tener con un cliente que describe el producto que desea obtener. Este procedimiento es característico de la metodología Scrum, en donde el desarrollo del proyecto comienza con una caracterización de las necesidades generales del cliente.

En segundo lugar, el presente proyecto de grado, al requerir una labor investigativa y de pruebas de diversos algoritmos de Criptografía de Curva Elíptica, puede favorecerse utilizando una metodología en donde las tareas de análisis y desarrollo se realicen de forma iterativa y evolutiva.

En tercer lugar, siendo este un proyecto unipersonal en su totalidad, el empleo de una metodología ágil facilita y ameniza la organización y desarrollo de todas las tareas involucradas, debido a que las metodologías ágiles para el desarrollo de proyectos de software promueven el trabajo de equipos o personas de manera auto-organizada, efectuando tomas de decisiones con efectos a corto plazo.

Finalmente, se busca emplear una metodología de carácter realista y de ámplio uso en los equipos laborales de desarrollo de software de la actualidad. Las tareas de detallar los requisitos del producto, y planificar junto con el cliente y de forma iterativa, qué se va a desarrollar en cada etapa, es un modelo de trabajo ágil, realista, y que se ajusta al modelo de trabajo que propone la metodología **Scrum**.

4.1.1. Metodología Ágil de Desarrollo

El término "Ágil" aplicado al desarrollo de software nace en una reunión celebarada en febrero de 2001 en Utah, Estados Unidos, entre 17 expertos de la industria del software, incluyendo algunos creadores o impulsores de metodologías de software. El objeto de esta reunión fue tratar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se prentendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación generada en cada una de las actividades desarrolladas. Partiendo de esto, se desarrolló el Manifiesto Ágil, un documento que resume la filosofía de desarrollo de proyectos de software "Ágiles"[1].

4.1.1.1. El Manifiesto Ágil

El Manifiesto enumera los principales valores del desarrollo Ágil[2]:

- Individuos e interacciones: La gente es el principal factor de éxito de un proyecto de software. Es más importante construir un buen equipo que construir el entorno.

[1] José H. Canós, Patricio Letelier y María Carmen Penandés. 2003. *Metodologías Ágiles para el desarrollo de software*.

[2] José H. Canós, Patricio Letelier y María Carmen Penandés. 2003. *Metodologías Ágiles para el desarrollo de software*.

- Desarrollar software que funciona: Nos enfocamos en no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Los documentos deben ser cortos y concentrarse en lo fundamental.
- Colaboración con el cliente: Se supone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- Responder ante el cambio: La habilidad de responder a los cambios que pueden surgir a lo largo del proyecto determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible.

4.1.2. Scrum

Scrum es una metodología ágil y flexible, aplicable a la gestión de desarrollo de software. Mediante esta metodología se alienta al trabajo en equipo y al proceso de desarrollo de software en forma iterativa.[3]

Originalmente, SCRUM fue ideado para la creación de productos tecnológicos, pero es igualmente aplicable a proyectos donde se trabaja con requisitos inestables y que requieren flexibilidad y rápida respuesta

Un proyecto desarrollado empleando SCRUM contempla los siguientes aspectos:

- Desarrollo de software en etapas incrementales
- Entregas de software terminado
- Adaptabilidad y facilidad de aprendizaje
- Basado en la experiencia del equipo de trabajo
- Trabajo con una metodología de bajo riesgo y costo

4.1.2.1. Elementos en un proyecto Scrum

Roles del equipo El **product owner** es la persona o entidad responsable de analizar las características enunciadas en el product backlog, y decidir cuáles serán incluidas en el proyecto, asignándoles prioridades. Es la parte del proyecto encargada de velar por los intereses de clientes y usuarios finales.

El **scrum master** es la parte que lidera al equipo de desarrollo, y se encarga de facilitarles todo lo que necesiten para que ejecuten el trabajo de forma óptima. También es el encargado de realizar las entrevistas para coordinar esfuerzos. Su rol no se enfoca en la posesión de autoridad administrativa, sino de efectuar un esfuerzo de sinergia entre las diferentes partes del equipo de desarrollo.

Los **desarrolladores y testers** escriben el código y se aseguran de que el trabajo producido funcione como fue solicitado en un principio.

[3]Ken Schwaber. 1995. “SCRUM Development Process”. En: *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, págs. 117-134.

Product Backlog En Scrum, las características fundamentales del producto a generar son conocidas como **user stories** (historias de usuario). Pueden ser producidas por clientes, ejecutivos, o incluso miembros del equipo de desarrollo, y son escritas desde la perspectiva del usuario final. La colección de todas estas historias es lo que se denomina **product backlog**. El **product owner** (propietario del producto) es la parte que representa a los usuarios y clientes del producto, y decide qué características del product backlog serán consideradas para el proyecto.

Release backlog El objetivo de un determinado **release** es el de entregar una porción del product backlog, conocida como **release backlog**. Después de identificar qué características serán incluidas en un determinado release, estas características pasan a formar parte del release backlog.

Sprints Un **sprint** es un hito de corta duración, en donde se toma una porción manejable del release backlog y la desarrolla hasta convertirla en un elemento funcional y entregable. Los sprints generalmente duran entre 5 a 30 días.

Al final de cada sprint, se debería obtener una porción de producto totalmente testeada y funcional, completada al 100

Burndowns El progreso del trabajo del equipo es seguido utilizando un **burndown chart**. El burndown chart es una herramienta para visualizar el progreso del proyecto, y provee una medición día a día del trabajo restante de un determinado sprint o release.

4.1.2.2. Fases de un proyecto Scrum

Scrum es una metodología ágil que emplea una estructura incremental basada en iteraciones y revisiones. El ciclo de vida Scrum se compone de tres fases:

- **Pre-game**: La etapa Pre-Game involucra dos actividades fundamentales: Planeamiento y Diseño de Arquitectura de Alto Nivel. Durante esta etapa se define el Product Backlog y se subdivide en unidades de trabajo que serán abarcadas en cada sprint durante la etapa de Game. A su vez, se definen las herramientas, frameworks y arquitectura de alto nivel del proyecto de software que se realizará.
- **Game**: La etapa Game usualmente es llamada la "etapa de desarrollo". Es la etapa en donde se realiza el desarrollo en ciclos iterativos (o Sprints) del proyecto de software que se desea crear.
- **Post-Game**: Durante la etapa de Post-Game todos los desarrollos de software están finalizados, y se procede a preparar para realizar pruebas, documentaciones, entrenamiento y despliegues.

4.1.3. Análisis y Diseño: UWE - UML-based Web Engineering

Para la realización del análisis y diseño del presente proyecto, emplearemos las herramientas de modelado provistas por la metodología UWE (UML-based Web Engineering). Con el objetivo de realizar una breve referencia a esta metodología, en esta sección haremos referencia a los conceptos detallados en el sitio web "UWE – UML-based Web Engineering" publicado por *Research Unit of Programming and Software Engineering* de la Universidad de Múnich[4].

[4]LMU – Ludwig-Maximilians-Universität München, Research Unit of Programming and Software Engineering. 2016. *UWE – UML-based Web Engineering*. URL: http://uwe.pst.ifi.lmu.de/aboutUwe.html (visitado 27-09-2017).

UWE es una herramienta de Ingeniería de Software del dominio del desarrollo Web, empleada para cubrir el ciclo completo del desarrollo de aplicaciones Web. El foco principal de la metodología UWE es proveer:

- Un lenguaje de modelado basado en UML
- Una metodología dirigida al modelo
- Una herramienta de soporte para la ingeniería de aplicaciones Web.

La notación de UWE es una extensión liviana de UML (Univied Modeling Language), que provee un perfil de UML apto para ser utilizado en el diseño de aplicaciones Web. UWE utiliza notación UML pura y diagramas UML siempre que sea posible para el análisis y diseño de aplicaciones Web. Para aspectos específicos del desarrollo Web, tal como nodos, links y estructuras HTTP, UWE incluye estereotipos, valores etiquetados y restricciones definidas para el modelado de dichos elementos. UWE es empleado para diseñar aspectos de navegación, presentación y procesos de negocio en aplicaciones Web.

Los modelos que se emplean en UWE son:

- Modelo de Casos de Uso, para capturar los requisitos del sistema.
- Modelo Conceptual para el contenido (modelo de dominio).
- Modelo de Navegación
- Modelo de Presentación

4.1.3.1. Análisis de Requerimientos con Casos de Uso

Un Modelo de Casos de Uso describe un trozo de comportamiento de la aplicación sin revelar su estructura interna.

4.1.3.2. Modelo Conceptual

El objetivo del diseño conceptual es construir un modelo de la aplicación considerando los requisitos reflejados en los casos de uso. Su resultado es un diagrama de clases de dominio. Los elementos característicos de este modelo son las clases y sus asociaciones.

4.1.3.3. Modelo de Navegación

El Modelo de Navegación conforma un paso de gran importancia para una aplicación Web, y describe la estructura de elementos a ser visitados mediante la navegación a través de la aplicación.

4.1.3.4. Modelo de Presentación

El Modelo de Presentación consiste en un conjunto de vistas que muestran el contenido y la estructura de los diferentes nodos descriptos en el Modelo de Navegación.

4.2. Planificación del desarrollo del proyecto

Como fue mencionado anteriormente, el proyecto fue desarrollado empleando la metodología ágil Scrum, complementada con la metodología UWE para el análisis y desarrollo del producto objeto de este proyecto.

4.2.1. Pre-Game

La etapa que abre un proyecto empleando metodología Scrum, es la etapa de "Pre-Game". Durante esta primera etapa se realiza una planificación generalizada de los aspectos que involucran el desarrollo del proyecto.

4.2.1.1. Product Backlog

En primer lugar, detallamos el Product Backlog, en donde se listan los requerimientos y características finales del proyecto.

ID	Descripción	Módulo	Prioridad
#01	Plataforma web de generación y verificación de firmas electrónicas de uso interno en una organización	Producto general	Alta
#02	Implementación de una PKI con esquema de Criptografía Delegada	Producto general	Alta
#03	Sistema Web con estructura Cliente - Servidor	Producto general	Alta
#04	Todas las operaciones criptográficas implementadas en una Librería criptográfica modular (independiente del sistema)	Servidor	Alta
#05	Esquema criptográfico basado en Criptografía de Curva Elíptica	Servidor	Alta
#06	Esquema criptográfico basado en Criptografía de Curva Elíptica	Servidor	Alta
#07	Ingreso al sistema mediante autenticación de dos pasos (contraseña y clave TOTP)	Servidor	Alta
#08	Servidor es un WebAPI RESTful	Servidor	Alta
#09	Aplicación Cliente es una aplicación JavaScript de navegador web	Servidor	Alta

Cuadro 4.1: Product Backlog

4.2.1.2. Arquitectura del sistema / Diseño de alto nivel

El sistema será desarrollado siguiendo una arquitectura "Cliente - Servidor". El servidor contará con un Web API, encargado de responder a todas las llamadas desde las plataformas cliente. A su vez, contará con una librería criptográfica independiente, que se utilizará para realizar todas las operaciones criptográficas del sistema. Los fundamentos de la elección de la arquitectura, así como del desarrollo de una librería criptográfica independiente, serán analizados en profundidad en la sección 4.3.1 (página 46).

4.2.1.3. Tecnologías de desarrollo

La construcción del proyecto se realizará, como mencionamos previamente, en una arquitectura por capas, de cliente-servidor.

WebAPI del Sistema de Firma Electrónica - Lenguaje y Framework El WebAPI del Sistema de Firma Electrónica será programado en el lenguage C#, utilizando el framework ASP.NET Core. Este framework, gratuito y de código abierto, permite compilar aplicaciones web tanto para ser ejecutado en plataformas Windows como Linux. La elección de C#/Asp.Net Core como tecnología para implementar el sistema se debe a que es un framework ampliamente utilizado en

el mercado, y por ende, posee una gran comunidad para brindar soporte. Finalmente, el hecho de que C# sea un lenguage compilado puede ser considerado un factor importante de seguridad: Intrusos no podrán tener acceso al código fuente del sistema para buscar posibles falencias de seguridad.

Cliente Front-End - Lenguaje y Framework El cliente Front-End es la interfaz con la que se comunica el usuario, a través de cualquier Explorador Web del que el usuario disponga. Será programado AngularJS, un framework JavaScript utilizado para el desarrollo de aplicaciones web denominadas "de una sola página". Esto es, aplicaciones JavaScript que se ejecutan en el explorador web del cliente. De esta forma, se minimiza el tráfico entre el cliente y el servidor al descargar de una sola vez la aplicación cliente, y realizar comunicaciones periódicas con el WebAPI del servidor a medida que el usuario realice operaciones.

Librería Criptográfica - Lenguaje y Framework La librería criptográfica será desarrollada empleando C# .NET Core, generando una librería que será empleada por el Servidor WebAPI para la realización de todas las operaciones criptográficas: Generación de pares de claves (pública y privada), encriptación y desencriptación de clave privada, comprobación de códigos TOTP, generación de CSRs, generación y verificación de firmas digitales. La elección de estas tecnologías para el desarrollo de la librería criptográfica se fundamenta de la misma manera que para el WebAPI previamente mencionado.

Base de Datos La base de datos del Sistema Interno de Firma Electrónica será desarrollada utilizando el motor de base de datos MySQL. Éste es un motor de base de datos gratuito y de código abierto. Sus implementaciones de seguridad como encriptación de datos y sistema de control de acceso basado en permisos y sistema de contraseñas, son fundamento para la elección de esta tecnología para almacenar los datos del sistema.

4.2.2. Game

La segunda fase del proyecto fue desarrollada en cuatro sprints. Cada uno de estos sprints, correspondientes a cada uno de los componentes principales del sistema objeto:

- **Sprint 1**: Diseño de implementación del sistema
- **Sprint 1**: Implementación de la Librería criptográfica
- **Sprint 2**: Implementación del Servidor WebAPI
- **Sprint 3**: Implementación del Cliente Web Front-End

Durante cada sprint se construyeron los diferentes modelos de la metodología UWE para generar una especificación a partir de la cual, luego, se implementarán los componentes de software empleando las tecnologías previamente mencionadas.

4.2.2.1. Sprint 1: Diseño de implementación del sistema

		Sprint	Inicio	Duración
		1	-	1 día (8 hs.)
ID	**Tarea**	**Tipo**	**Duración**	**Estado**
1.1	Planificación del sprint	Planificación	1 hora	Finalizado
1.2	Análisis de requerimientos del Product Backlog	Planificación	2 horas	Finalizado
1.3	Documentación de Casos de Uso del sistema	Desarrollo	3 horas	Finalizado
1.4	Construcción del diagrama de implementación del sistema	Desarrollo	2 horas	Finalizado

Cuadro 4.2: Planificación del Sprint 1

4.2.2.2. Sprint 2: Implementación de la librería criptográfica

		Sprint	Inicio	Duración
		1	-	15.75 días (126 hs.)
ID	**Tarea**	**Tipo**	**Duración**	**Estado**
2.1	Planificación del sprint	Planificación	2 horas	Finalizado
2.2	Análisis de requerimientos del Product Backlog para la Librería Criptográfica	Planificación	2 horas	Finalizado
2.3	Construcción del diagrama de clases de la librería	Desarrollo	2 horas	Finalizado
2.4	Implementación, prueba y selección de algoritmos de Sumado de Puntos de Curva Elíptica	Desarrollo	2 días	Finalizado
2.5	Implementación, prueba y selección de algoritmos de Duplicado Puntos de Curva Elíptica	Desarrollo	2 días	Finalizado
2.6	Implementación, prueba y selección de algoritmos de Reducción Modular	Desarrollo	2 días	Finalizado
2.7	Implementación, prueba y selección de algoritmos de Inversión Modular Multiplicativa	Desarrollo	2 días	Finalizado
2.8	Desarrollo de la librería	Desarrollo	5 días	Finalizado
2.9	Testing de la librería	Desarrollo	2 días	Finalizado

Cuadro 4.3: Planificación del Sprint 2

4.2.2.3. Sprint 3: Servidor

		Sprint	Inicio	Duración
		2	-	9.25 días (74 horas)
ID	**Tarea**	**Tipo**	**Duración**	**Estado**
3.1	Planificación del sprint	Planificación	2 horas	Finalizado
3.2	Análisis de requerimientos del Product Backlog para el servidor del sistema	Planificación	2 horas	Finalizado
3.3	Construcción del diagrama entidad-relación	Desarrollo	2 horas	Finalizado
3.4	Desarrollo del esquema de base de datos en MySQL	Desarrollo	4 horas	Finalizado
3.5	Construcción de diagramas de secuencia del servidor	Desarrollo	1 día	Finalizado
3.6	Desarrollo del servidor WebAPI	Desarrollo	5 días	Finalizado
3.7	Testing del servidor WebAPI	Desarrollo	2 días	Finalizado

Cuadro 4.4: Planificación del Sprint 3

4.2.2.4. Sprint 4: Cliente Web Front-End

		Sprint	Inicio	Duración
		3	-	8.75 días (70 horas)
ID	**Tarea**	**Tipo**	**Duración**	**Estado**
4.1	Planificación del sprint	Planificación	2 horas	Finalizado
4.2	Análisis de requerimientos del Product Backlog para el Cliente Front-End	Planificación	2 horas	Finalizado
4.3	Análisis de requerimientos con casos de uso	Desarrollo	2 horas	Finalizado
4.4	Análisis de requerimientos con especificación de casos de uso	Desarrollo	2 horas	Finalizado
4.5	Construcción del modelo de navegación	Desarrollo	2 horas	Finalizado
4.6	Construcción del modelo de presentación	Desarrollo	4 horas	Finalizado
4.7	Desarrollo del Cliente Front-End	Desarrollo	5 días	Finalizado
4.8	Testing del Cliente Front-End	Desarrollo	2 días	Finalizado

Cuadro 4.5: Planificación del Sprint 4

4.2.3. Post-Game

Durante la fase Post-Game del proyecto se realizaron actividades de prueba y de preparación del sistema para el despliegue.

4.2.3.1. Pruebas de software

Se efectuaron dos tipos de pruebas en los diferentes elementos del sistema: Pruebas de Unidad, en el Servidor WebAPI y la Librería Criptográfica, y Pruebas de Navegación para el Cliente Front-End.

Las pruebas de unidad son de gran utilidad para verificar el correcto funcionamiento en fragmentos de código donde conocemos los resultados esperables. Por esta razón, las empleamos

para verificar el correcto funcionamiento en la Librería Criptográfica y en el Servidor WebAPI. En la Sub-Librería de Operaciones Criptográficas se realizaron pruebas de unidad en los sobre las siguientes funciones:

- FirmaElectronica.GenerarFirma
- FirmaElectronica.VerificarFirma

Por otra parte, en la Sub-Librería de Operaciones Aritméticas se realizaron pruebas de unidad en las siguientes funciones:

- Punto + Punto (Suma de puntos)
- 2 * Punto (Duplicación de puntos)
- k * Punto (Multiplicación de un punto por un entero)
- KeyPair.GenerarNuevo

Pruebas de Unidad en Servidor WebAPI Se efectuaron pruebas de unidad en cada "End-Point" del Servidor WebAPI, asegurando que las respuestas entregadas sean las esperadas definidas en cada prueba.

Pruebas de Navegación en Cliente Front-End Las pruebas de navegación consisten en verificar que toda la aplicación web del lado del Cliente funcione de manera adecuada, poniendo atención en los siguientes puntos:

- Todos los elementos de navegación (botones, links) deberán presentar la página web que indican textualmente.
- Cualquier acción prohibitiva (debido a intentos de acceso del usuario a opciones no permitidas) deberá producir un mensaje de error correspondiente.
- En caso de que expire la sesión del usuario (debido a inactividad o vencimiento del token) deberá hacer que se presente la pantalla de login.

4.2.3.2. Pruebas de seguridad web

La seguridad del Sistema de Firma Electrónica es un factor crítico para el éxito del proyecto. Por esta razón, es imperativo realizar pruebas de seguridad "web" sobre el Servidor WebAPI y el Cliente Front-End para garantizar que la implementación no presenta vulnerabilidades ante diferentes tipos de ataques. Para este propósito, empleamos la especificación más reciente provista por el proyecto Open Web Application Security Project: ***OWASP Top 10 - 2017***[5].

"OWASP Top 10" es un documento de referencia dedicado a la seguridad web publicado por Open Web Application Security Project. En este, se especifican los factores de riesgo más importantes a tener en cuenta en implementaciones web, consensuadas por expertos miembros del proyecto. En la última versión publicada del documento (OWASP Top 10 - 2017 Release Candidate) se especifican los siguientes riesgos de seguridad:

- A1 – Inyección

[5]The Open Web Application Security Project. 2017. *OWASP Top 10 - 2017 Release Candidate - The Ten Most Critical Web Application Security Risks*.

- A2 – Pérdida de Autenticación y Gestión de Sesiones
- A3 – Secuencia de Comandos en Sitios Cruzados (XSS)
- A4 – Pérdida de Control de Acceso
- A5 – Configuracion de Seguridad Incorrecta
- A6 – Exposición de Datos Sensibles
- A7 – Protección contra Ataques Insuficiente
- A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)
- A9 – Uso de Componentes con Vulnerabilidades Conocidas
- A10 – APIs desprotegidas

A partir de estos diez puntos críticos de seguridad web a tener en cuenta, se efectuaron las siguientes verificaciones:

A1 – Inyección Se verificó que ninguno de los End-Points del Servidor WebAPI emplee ejecuciones directas del intérprete del motor de base de datos (MySQL), y que los accesos a bases de datos sean realizados empleando un framework ORM (Object-Relation Mapping) seguro: EntityFramework. De esta manera, se garantiza que no se podrán realizar inyecciones SQL que puedan provocar accesos no permitidos a datos.

A2 – Pérdida de Autenticación y Gestión de Sesiones Este punto crítico de seguridad hace referencia a la correcta implementación de los sistemas de autenticación y manejo de sesiones. El Motor de Administración de Autenticación y Sesiones que se emplea en el presente proyecto es *ASP.NET Core Identity*. Se verificó que la versión empleada en el Servidor WebAPI no presente fallas de seguridad conocidas.

A3 – Secuencia de Comandos en Sitios Cruzados (XSS) Se verificó que ningún contenido HTML mostrado en el Cliente Front-End provenga de datos crudos ingresados a través del Servidor WebAPI. Es decir, fue comprobado que sean sanitizados los datos ingresados a través de todos los formularios del Sistema de Firma Electrónica, previo a ser almacenados o mostrados en el Cliente Front-End.

A4 – Pérdida de Control de Acceso Se verificó que cada End-Point del Servidor WebAPI pueda ser accedido únicamente por los usuarios del rol correspondiente. Es decir, que no haya accesos no autorizados a funciones del sistema por usuarios que no tengan los permisos correspondientes.

A5 – Configuracion de Seguridad Incorrecta En referencia a las secciones 4.3.7.2 y 4.3.7.4, los accesos a base de datos y al Servidor WebAPI deben estar completamente revisadas para evitar cualquier acceso intrusivo y uso no esperado de los componentes del sistema.

Fueron examinadas las configuraciones de seguridad del Servidor WebAPI y de la implementación del esquema de base de datos, verificando que no existan factores que puedan comprometer la integridad del Sistema de Firma Electrónica o de los datos almacenados por el mismo.

Tal como se indica en la sección 4.3.7.2, la base de datos fue verificada para ser accedida únicamente de forma local por el Servidor WebAPI, empleando las credenciales creadas para tal propósito.

Por otra parte, en referencia a la sección 4.3.7.4, el Servidor WebAPI fue verificado para que todos sus End-Points únicamente respondan a llamados efectuados empleando el protocolo SSL.

A6 – Exposición de Datos Sensibles En el Sistema de Firma Electrónica, consideramos ”datos sensibles” inherentes a la naturaleza del presente proyecto a:

- Los pares de claves encriptados almacenados en base de datos.
- Las credenciales de acceso de los usuarios.

Se verificó que los pares de claves y credenciales de acceso de los usuarios del Sistema de Firma Electrónica sean debidamente encriptados en la base de datos, de manera que resulte teóricamente imposible que un intruso se apodere de esta información.

A7 – Protección contra Ataques Insuficiente Fue verificado que el Servidor WebAPI solamente exponga los End-Points diseñados y detallados en la documentación del Sistema de Firma Electrónica, de manera que las comunicaciones entre los usuarios y el servidor se efectúen únicamente en términos conocidos y esperados.

A su vez, se probó que el sistema genere correctamente “logs” donde se detallen accesos y accionar de los usuarios, para futuras auditorías.

A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF) Se verificó que todos los End-Points del Servidor WebAPI empleen el sistema ASP.NET Core AntiForgery. Este paquete cumple específicamente la función de evitar falsificaciones de peticiones en sitios cruzados.

A9 – Uso de Componentes con Vulnerabilidades Conocidas Fueron analizados todas las librerías y paquetes empleadas en el Sistema de Firma Electrónica para verificar que no existan vulnerabilidades conocidas en las versiones utilizadas.

A10 – APIs desprotegidas Se verificó que todos los End-Points del Servidor WebAPI funcionen únicamente empleando el protocolo SSL/TLS, de manera que toda comunicación con el Cliente Front-End se efectúe de forma encriptada y segura. Fue comprobado el correcto funcionamiento de los métodos de autenticación, y la protección de los End-Points frente a solicitudes con formatos erróneos, que pueden llegar a provocar errores en el funcionamiento del Servidor WebAPI.

4.3. Desarrollo del proyecto

4.3.1. Diseño de implementación del proyecto

El diseño de la implementación del proyecto puede verse detallado en el diagrama de la figura 4.1. Existen diferentes razones que motivan la implementación del proyecto en un esquema modularizado. En primer lugar, diferenciar un software servidor (que en el presente proyecto denominaremos “Web API”) y una aplicación web cliente permite que en algún futuro el sistema pueda agregar nuevas aplicaciones cliente, de diferentes plataformas (tal como aplicaciones móviles que consuman métodos del servidor, aplicaciones de escritorio para Windows, etc.).

La implementación de una librería criptográfica separada del Web API se fundamenta en cuestiones de seguridad a largo plazo del proyecto. Los esquemas de criptografía que actualmente emplea cualquier sistema informático, son seleccionados bajo la premisa de que actualmente no existe ninguna computadora con la capacidad computacional para desencriptar un mensaje en un período de tiempo que resulte práctico. Por ejemplo, en un esquema criptográfico que utiliza claves privadas de 256 bits (como el esquema criptográfico que plantea utilizar este proyecto), implica la existencia de 2256 claves privadas diferentes. Esto es, un número de 78 cifras. Tomando como ejemplo la red de Bitcoin (la red de computadoras y equipos de computación dedicados a descifrar hashes para obtener Bitcoins), ésta tiene una capacidad computacional de descifrar aproximadamente 3600 PetaHashes por segundo[6]. Si estimamos que el tiempo que la red de Bitcoin necesita para ejecutar una operación de criptografía de Curvas Elípticas es el mismo tiempo que requiere para descifrar hashes SHA-256, entonces el tiempo que se necesitaría para descifrar una clave privada de 256 bits es:

$$\frac{2^{256bits}}{36 * 10^{17} \frac{operaciones}{segundo}} * \frac{1 \text{ hora}}{3600 \text{ segundos}} * \frac{1 \text{ día}}{24 \text{ horas}} * \frac{1 \text{ año}}{365{,}25 \text{ días}}$$

$$= 1{,}019 * 10^{39} \text{ billones de años}$$

Este es un número ampliamente aceptable para los requerimientos de seguridad según las capacidades computacionales actuales. Pero se predice que aproximadamente en 2025 va a haberse conseguido desarrollar de manera funcional la computación cuántica[7], siendo éste un paradigma de computación ampliamente documentado como "capaz de romper los esquemas de criptografía modernos" (ciertos algoritmos, como los Algoritmos de Shor[8] demuestran que en muy poco tiempo pueden romperse encriptaciones RSA mediante computación cuántica). Es por esta razón que es de gran utilizad implementar una librería criptográfica independiente del sistema, que permita fácilmente migrar de esquema criptográfico manteniendo la misma plataforma de Firma Electrónica.

[6] Wuille 2017.

[7] Bela Bauer et al. 2016. "Hybrid quantum-classical approach to correlated materials". En: *Physical Review X* 6.3.

[8] Peter W Shor. 1999. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". En: *SIAM review* 41.2.

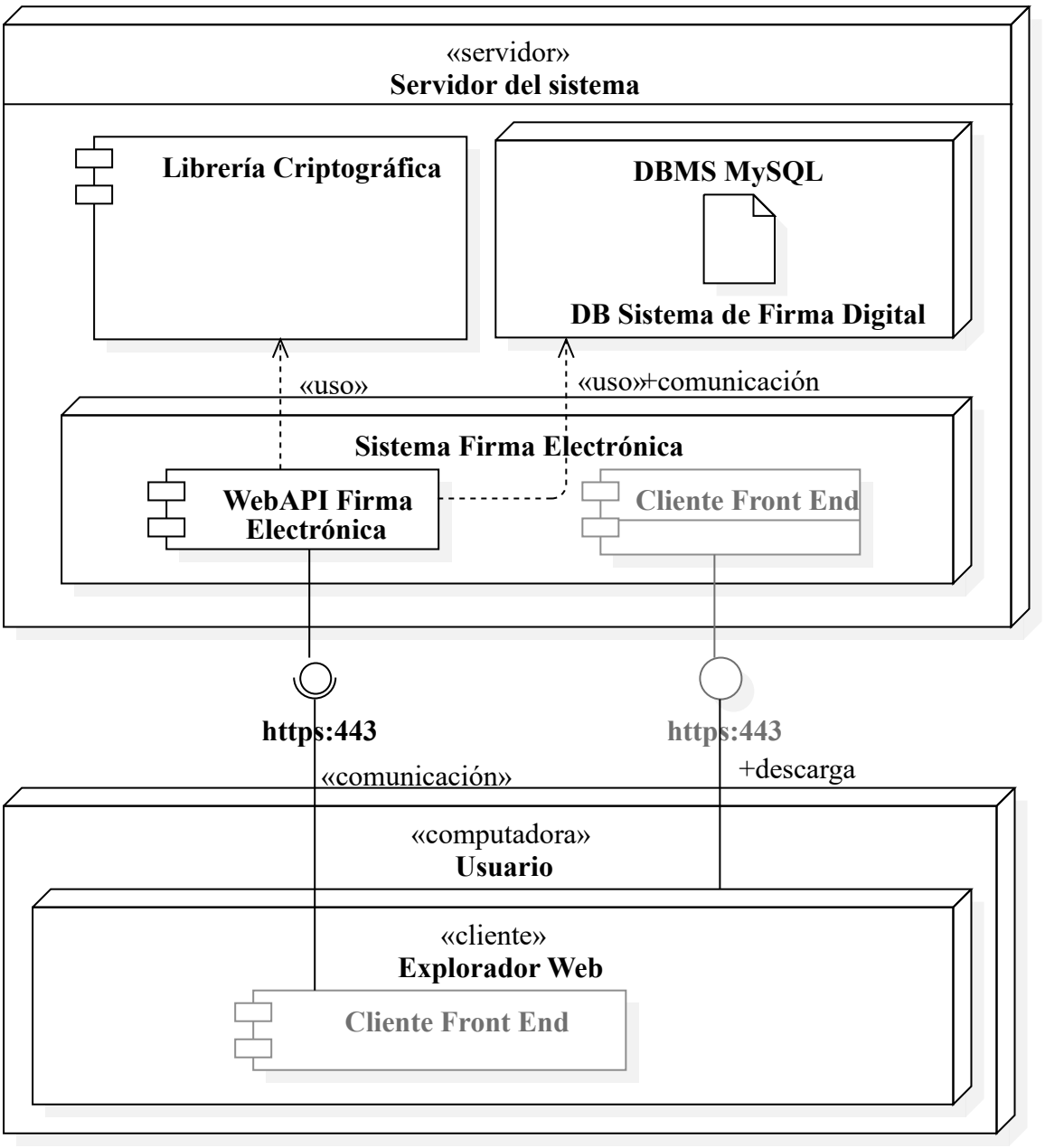

Figura 4.1: Diagrama de Despliegue

En la figura 4.1 podemos reconocer la plataforma “Servidor”, que es la computadora o equipo informático que hospedará el sistema del presente proyecto. En esta plataforma “Servidor” se encuentra el Sistema de Firma Electrónica, que consta de un WebAPI de Firma electrónica y el Cliente Front-End. Cuando un usuario, desde su computadora, ingresa al sistema mediante un explorador web, “descarga” el Cliente Front-End (comúnmente denominado Single Page Application, desarrollado en un framework Javascript). Todas las comunicaciones entre el Cliente Front-End y el Sistema de Firma Electrónica se realizan mediante el WebAPI.

El sistema emplea un esquema en una base de datos MySQL para dar persistencia a toda la información necesaria para su funcionamiento (en el Diagrama Entidad-Relación detallado en la figura 4.5 de página 62 se puede observar el diseño de la base de datos). Para efectuar todas las operaciones criptográficas, el sistema se vale de una Librería Criptográfica (en términos generales, una librería *.dll*) detallada y desarrollada en este proyecto.

Es importante remarcar que toda comunicación entre el usuario, utilizando su computadora, y el sistema, localizado en el servidor, se realizan empleando el protocolo ***https***. De tal modo, todas las comunicaciones se realizarán de forma segura.

4.3.2. Análisis de Requisitos

4.3.2.1. Diagrama de Casos de Uso

Para identificar correctamente las acciones que los usuarios realizarán empleando el sistema, presentamos el Diagrama de Casos de Uso de la figura 4.2.

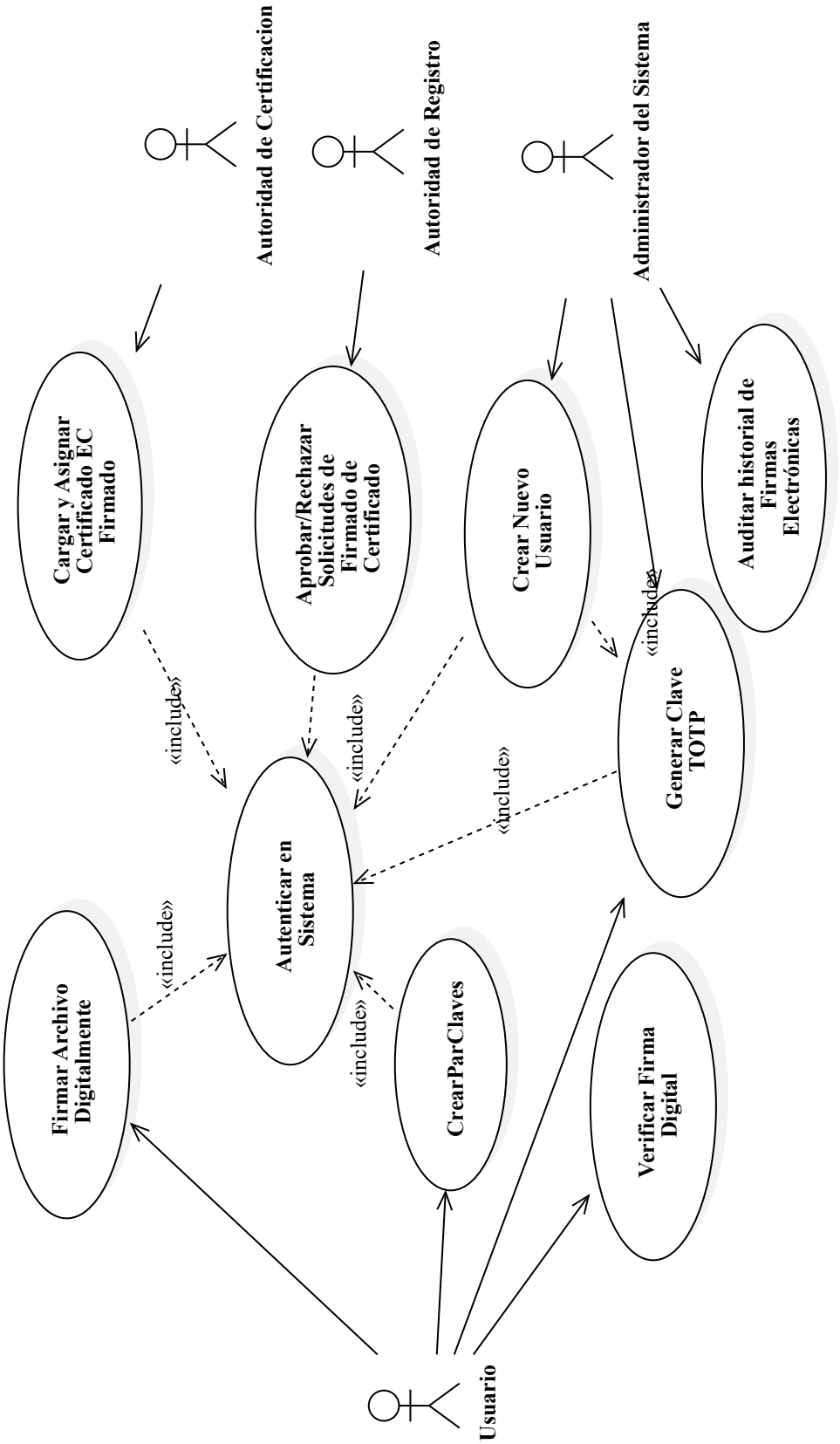

Figura 4.2: Diagrama de Casos de Uso

4.3.2.2. Especificación de Casos de Uso

CU	Autenticar Sistema
Actor	Usuario
Pre-Condiciones	Usuario sin autenticar en sistema
Post-Condiciones	Usuario autenticado en sistema
Secuencia	
Actor	**Sistema**
1. Usuario ingresa por primera vez al sistema web 2. Usuario ingresa nombre de usuario y password 5. Genera clave TOTP con celular 6. Ingresa clave TOTP en sistema	 3. Sistema valida usuario y password 4. Sistema solicita clave TOTP 7. Sistema valida clave TOTP 8. Sistema autentica usuario correctamente y presenta web principal
Escenario Alternativo	
Actor	**Sistema**
	3. Sistema valida usuario y password incorrecto 4. Sistema vuelve a solicitar usuario y contraseña 7. Sistema valida clave TOTP incorrecta 8. Sistema vuelve a solicitar clave TOTP

Cuadro 4.6: Caso de Uso: Autenticar Sistema

CU	Firmar Archivo Digitalmente
Actor	Usuario normal
Pre-Condiciones	Usuario autenticado en sistema
Post-Condiciones	Generada firma electrónica de archivo
Secuencia	
Actor	**Sistema**
1. Usuario selecciona opción para firmar archivo digitalmente 2. Usuario selecciona archivo a firmar digitalmente 3. Usuario selecciona la clave a emplear para el firmado 6. Usuario descarga firma electrónica	 4. Sistema genera firma electrónica 5. Sistema muestra link de descarga de la firma electrónica

Cuadro 4.7: Caso de Uso: Firmar Archivo Digitalmente

CU	Verificar Firma Electrónica
Actor	Usuario normal
Pre-Condiciones	Usuario autenticado en sistema
Post-Condiciones	Resultados de verificación obtenidos

Secuencia	
Actor	**Sistema**
1. Usuario selecciona opción para verificar firma electrónica 2. Usuario selecciona archivo a verificar 3. Usuario selecciona firma electrónica a verificar	4. Sistema verifica firma electrónica 5. Sistema entrega información y resultados de verificación

Cuadro 4.8: Caso de Uso: Verificar Firma Electrónica

CU	Generar nuevo par de claves
Actor	Usuario normal
Pre-Condiciones	Administrador autenticado en sistema
Post-Condiciones	CSR nuevo generado y descargado

Secuencia	
Actor	**Sistema**
1. Usuario selecciona opción “Crear nuevo par de claves” 2. Usuario completa los datos del formulario para la Solicitud de Firmado de Certificado 3. Usuario confirma el formulario, creando la solicitud.	4. Sistema confirma creación de nuevo par de claves y Solicitud de Firmado de Certificado
Escenario Alternativo	
Actor	**Sistema**
6. [Regresa a 2]	4. Servidor informa errores en datos cargados

Cuadro 4.9: Caso de Uso: Generar nuevo par de claves

CU	Verificar datos de Solicitud de Firmado de Certificado
Actor	Autoridad de Registro
Pre-Condiciones	Autoridad de Registro autenticado en sistema
Post-Condiciones	Solicitud de Firmado de Certificado aprobada/rechazada

Secuencia	
Actor	**Sistema**
1. AR selecciona selecciona Solicitud de Firmado de Clave para verificar 2. AR corrobora la veracidad de los datos ingresados 3. AR aprueba/rechaza la Solicitud de Firmado de Certificado	

Cuadro 4.10: Caso de Uso: Verificar datos de Solicitud de Firmado de Certificado

CU	Firmar y cargar certificado
Actor	Autoridad Certificante
Pre-Condiciones	Autoridad Certificante autenticado en sistema
Post-Condiciones	Certificado cargado en sistema y asignado a clave de usuario

Secuencia	
Actor	**Sistema**
1. AC selecciona selecciona Solicitud de Firmado de Clave para firmar 2. AC descarga la Solicitud de Firmado de Certificado 3. AC firma la Solicitud de Firmado de Certificado, generando certificado 5. AC carga el certificado generado en la página de la Solicitud de Firmado de Certificado correspondiente	6. Sistema informa operación realizada correctamente
Escenario Alternativo	
Actor	**Sistema**
7. [Regresa a 5]	6. Sistema informa operación realizada con errores

Cuadro 4.11: Caso de Uso: Firmar y cargar certificado

CU	Generar Clave TOTP
Actor	Administrador de Sistema
Pre-Condiciones	Administrador autenticado en sistema
Post-Condiciones	Dispositivo móvil del usuario con generador de códigos TOTP cargado

Secuencia	
Actor	**Sistema**
1. Administrador selecciona opción para generar claves TOTP	
2. Administrador selecciona usuario a asignar clave TOTP	
	3. Sistema genera nueva clave TOTP
4. Administrador carga clave TOTP en el dispositivo móvil del usuario	
5. Administrador genera código de prueba	
6. Administrador ingresa código de prueba	
	7. Sistema valida código de prueba correctamente
Escenario Alternativo	
Actor	**Sistema**
	7. Sistema valida código de prueba incorrectamente
8. [Regresa a 6]	

Cuadro 4.12: Caso de Uso: Generar Clave TOTP

CU	Crear Nuevo Usuario
Actor	Administrador de Sistema
Pre-Condiciones	Administrador autenticado en sistema
Post-Condiciones	Usuario creado

Secuencia	
Actor	**Sistema**
1. Administrador selecciona opción para crear usuario	
2. Administrador completa formulario con datos del usuario	
3. Administrador confirma datos de usuario	
	4. Sistema valida datos de usuario
	5. Sistema envía password autogenerado al email del usuario
	6. Sistema informa operación realizada con éxito
Escenario Alternativo	
Actor	**Sistema**
	4. Sistema valida datos de usuario con error
5. [Regresa a 2]	

Cuadro 4.13: Caso de Uso: Crear Nuevo Usuario

CU	Auditar historial de Firmas Electrónicas
Actor	Administrador del Sistema
Pre-Condiciones	Usuario autenticado en sistema
Post-Condiciones	Lista de Firmas Electrónicas generadas

Secuencia	
Actor	**Sistema**
1. Usuario selecciona opción para ver Historial de Firmas Electrónicas	

Cuadro 4.14: Caso de Uso: Auditar historial de Firmas Electrónicas

4.3.3. Librería Criptográfica

La librería criptográfica es el componente encargado de efectuar las operaciones inherentes a la generación y verificación de Firmas Electrónicas. Las clases y métodos contenidos en esta librería pueden ser agrupados en dos librerías internas. En primer lugar, una Sub-Librería de Operaciones Criptográficas que presenta las clases y métodos accedidos por la aplicación Servidor WebAPI para realizar las operaciones criptográficas. En segundo lugar, una Sub-Librería de Operaciones Aritméticas donde se implementan las clases y métodos del modelo matemático, empleados para realizar las operaciones criptográficas.

Antes de comenzar con esta implementación, debemos seleccionar un tamaño de claves (comúnmente denominado en inglés “key length”), y un esquema de hash. Siguiendo las recomendaciones publicadas por las siguientes agencias nacionales:

- Bundesamt für Sicherheit in der Informationstechnik (Oficina federal de seguridad de la información, Alemania)[9]
- National Institute of Standards and Technology (Instituto nacional de estándares y tecnología, Estados Unidos)[10]
- Agence nationale de la sécurité des systèmes d’information (Agencia nacional de seguridad de sistemas de información, Francia)[11]

Elegimos un tamaño de clave privada de curva elíptica de 256 bits, y el esquema de hash SHA-256. Las tres agencias coinciden en cuanto a la validez de las claves privadas de 256 bits y el algoritmo de hash SHA-256, en cuanto a que son considerados seguros previendo la evolución de los sistemas de información en un período mayor a 15 años (se supone que los esquemas elegidos pueden seguir siendo utilizados con seguridad después de 2030)[12][13][14].

Se utilizó un Diagrama de Clases para describir las clases (junto a sus métodos y propiedades) que componen esta Librería Criptográfica.

La implementación completa de esta librería, en lenguaje C#, puede observarse en el apéndice B del presente Proyecto de Grado.

[9] Bundesamt für Sicherheit in der Informationstechnik 2017.
[10] NIST (National Institute of Standards and Technology) 2016.
[11] Agence nationale de la sécurité des systèmes d’information 2014.
[12] Bundesamt für Sicherheit in der Informationstechnik 2017.
[13] NIST (National Institute of Standards and Technology) 2016.
[14] Agence nationale de la sécurité des systèmes d’information 2014.

4.3.3.1. Sub-Librería de Operaciones Criptográficas

La Sub-librería de Operaciones Criptográficas es la que provee los métodos empleados por el Servidor WebAPI. Contextualizándolo, podemos decir que es la parte "visible" de la Librería Criptográfica. En esta sub-librería debemos implementar los métodos para generar las firmas electrónicas de los archivos solicitados por los usuarios, y las verificaciones.

CertificateSigningRequest CertificateSigningRequest es la clase estática que provee el método para generar una nueva Solicitud de Firmado de Certificado (CSR) a partir del par de claves y de los datos ingresados por el usuario. La Solicitud de Firmado de Certificado generada es la que deberá descargar la Autoridad Certificante para generar el certificado correspondiente.

KeyPair La clase KeyPair es utilizada para instanciar los pares claves (pública y privada), o generar un nuevo par. Este objeto es utilizado para la generación de CSRs y de firmas electrónicas, o para la validación de las mismas.

FirmaElectronica Esta clase estática posee los métodos para generar y validar firmas electrónicas.

DatosSeguridad Esca clase representa a los datos de seguridad propios de cada usuario, empleados para la encriptación y desencriptación de sus claves privadas. El funcionamiento del diseño se encuentra descripto en la sección 4.3.6.5.

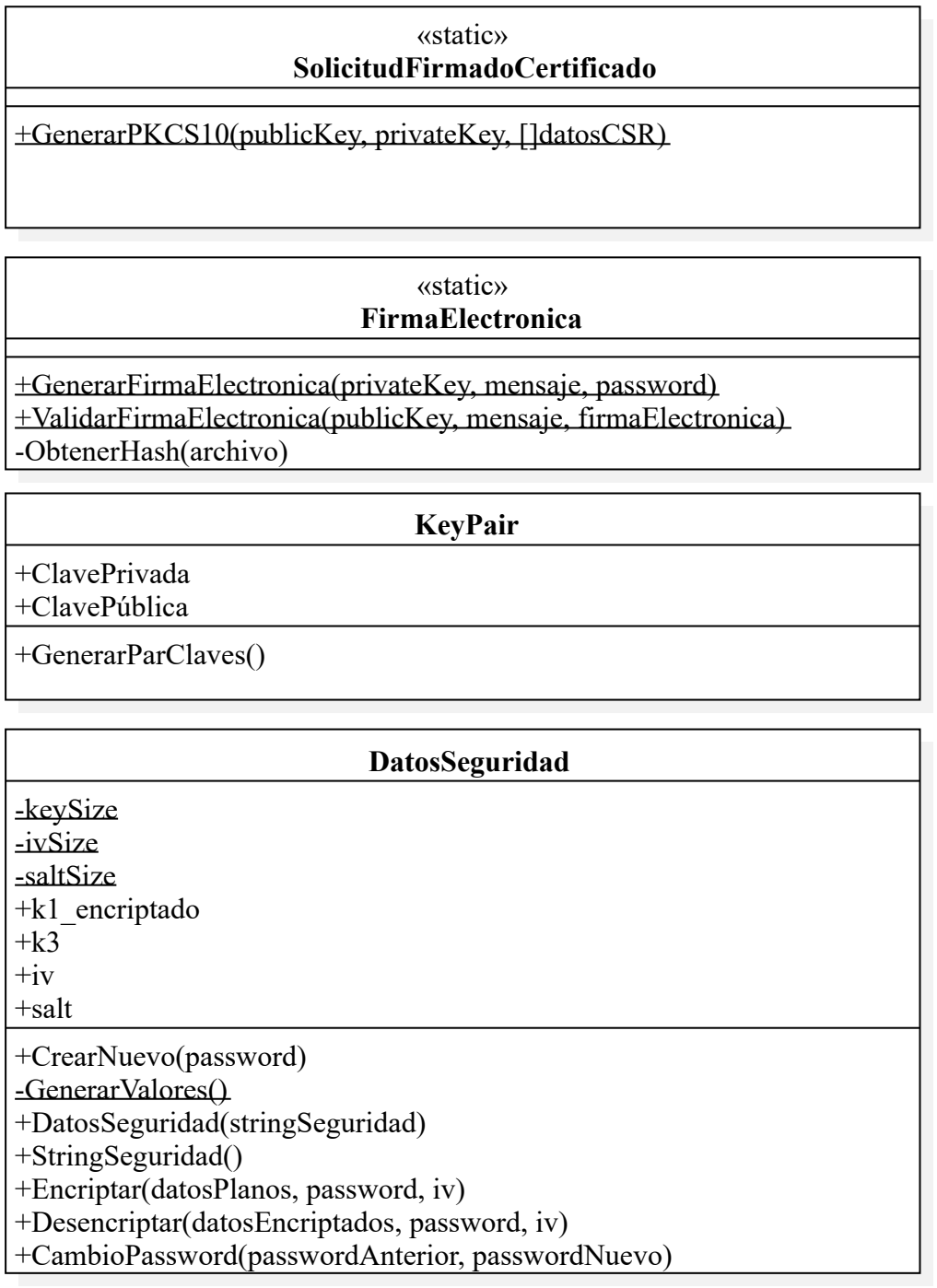

Figura 4.3: Diagrama de Clases de la Sub-Librería de Operaciones Criptográficas

4.3.3.2. Sub-Librería de Operaciones Aritméticas de Curva Elíptica

La SubLibrería de Operaciones Aritméticas es la librería que implementa los modelos matemáticos y operaciones en los que se basa la Criptografía de Curvas Elípticas. Tal como se detalló en la sección 2.3.1, las operaciones de suma, duplicación y multiplicación de puntos en una determinada curva elíptica son operaciones núcleo del esquema de criptografía

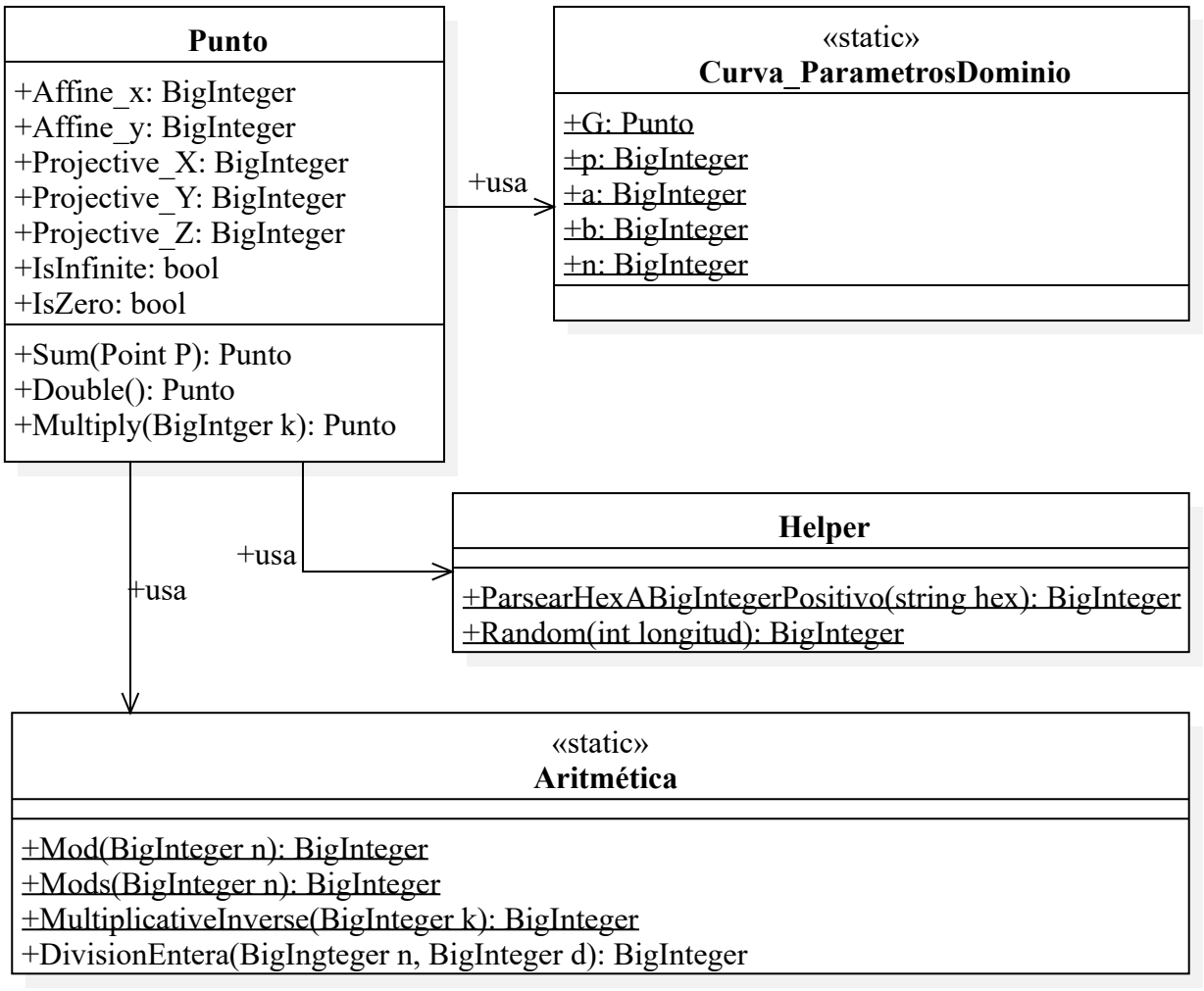

Figura 4.4: Diagrama de Clases de la Sub-Librería de Operaciones Aritméticas

Para la implementación, debemos elegir una curva elíptica (con sus parámetros dominio), y elegir los algoritmos a implementar para las operaciones aritméticas fundamentales.

La curva elíptica elegida es la SECP256k1, documentada en el ensayo del año 2010 publicado por Certicom Research "SEC 2: Recommended Elliptic Curve Domain Parameters"[15]. La curva SECP256k1 es una curva elíptica que ganó gran popularidad al comenzar a ser utilizada por el sistema de moneda electrónica Bitcoin. Esta curva se caracteriza por haber sido construida empleando métodos no aleatorios (al contrario de las curvas elípticas propuestas por el "National Institute of Standards and Technology" de Estados Unidos), y que da la ventaja de que pueden implementarse operaciones sobre sus puntos con diversas optimizaciones.

Los parámetros a implementar de esta curva elíptica son los siguientes:

```
p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   FFFFFFFE FFFFFC2F

a = 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000

b = 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00000007
```

[15] Certicom Research 2010.

```
G = 04
    79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59
       F2815B 16F81798
    483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9
       C47D08F FB10D4B8

n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B
   BFD25E8C D0364141

h = 01
```

Luego de implementar la curva con sus parámetros dominio, debemos continuar con las tres operaciones aritméticas fundamentales, realizables sobre los puntos de la curva elíptica: Suma, duplicación y multiplicación. Las tres son operaciones de gran complejidad, y es indispensable buscar los algoritmos óptimos para efectuarlas minimizando el uso del procesador y memoria, y optimizando el tiempo que requiren para completarse.

Los algoritmos de suma y duplicación elegidos son los planteados por Bernstein y Lange, en su ensayo "Faster addition and doubling on elliptic curves"[16]. Se eligieron los planteos empleando Coordenadas Proyectivas Jacobianas. Los algoritmos que emplearemos son los siguientes:

Operación de Suma

Algoritmo 4.1: Suma de Puntos Bernstein–Lange (2007)

Entrada: Punto A $= (X1, Y1, Z1)$, Punto B $= (X2, Y2, Z2)$
Salida: $(X3, Y3, Z3) = A + B$

1 $z1z1 = Z1^2$;
2 $z2z2 = Z2^2$;
3 $u1 = X1 * z2z2$;
4 $u2 = X2 * z1z1$;
5 $s1 = Y1 * Z2 * z2z2$;
6 $s2 = Y2 * Z1 * z1z1$;
7 $h = u2 - u1$;
8 $i = (2 * h)^2$;
9 $j = h * i$;
10 $r = 2 * (s2 - s1)$;
11 $v = u1 * i$;
12 $X3 = r^2 - j - s * v$;
13 $Y3 = r * (v - X3) - 2 * s1 * j$;
14 $Z3 = ((Z1 + Z2)^2 - z1z1 - z2z2) * h$;

[16] Bernstein y Lange 2007.

Operación de Duplicación

Algoritmo 4.2: Duplicación de Puntos Bernstein–Lange (2007)

Entrada: Punto A $= (X1, Y1, Z1)$, parámetro a de la curva elíptica
Salida: $(X3, Y3, Z3) = 2 * A$

1 $xx = X1^2$;
2 $yy = Y1^2$;
3 $yyyy = yy^2$;
4 $zz = Z1^2$;
5 $s = 2 * ((X1 + yy)^2 - xx - yyyy)$;
6 $m = 3 * xx + a * zz^2$;
7 $t = m^2 - 2 * s$;
8 $X3 = t$;
9 $Y3 = m * (s - t) - 8 * yyyy$;
10 $Z3 = (Y1 + Z1)^2 - yy - zz$;

Las implementaciones en lenguaje C# de estos dos algoritmos pueden observarse en los apéndices B.2.1.3 y B.2.1.4.

Operación de Multiplicación

La operación de multiplicación de un escalar por un punto de curva elíptica es la más compleja e importante de implementar. Su optimalidad va a determinar la velocidad con la que se van a efectuar operaciones como generación y verificación de firmas electrónicas, y la carga que van a significar para el servidor que hospede el sistema. Existen numerosos algoritmos propuestos con diferentes características. Por ejemplo, hay algoritmos de multiplicación óptimos para ser implementados en hardware y otros para ser implementados utilizando un lenguaje de programación.

Para la elección de un algoritmo de multiplicación, fueron efectuadas pruebas sobre los algoritmos detallados por Hankerson, Menezes y Vanstone en su libro “Guide to Elliptic Curve Cryptography”[17]:

- Right-to-left binary method
- Left-to-right binary method
- Binary NAF method
- Window NAF method
- Fixed-base windowing method

Se generaron 100 números aleatorios utilizando como máximo el parámetro “n” de la curva elíptica Secp256k1. Luego se efectuaron multiplicaciones escalares del punto base “G” de la curva elíptica elegida por los números aleatorios generados. El tiempo en el que se realizaron las 100 multiplicaciones, empleando cada uno de los algoritmos es el siguiente:

[17] Hankerson, Menezes y Vanstone 2010.

Algoritmo de multiplicación	Tiempo
Binary NAF Point Multiplication	1116,0089 ms
Left-to-right binary method	1248,9561 ms
Right-to-left binary method	1404,3362 ms
Window NAF method	1078,9081 ms
Fixed-base windowing method (window width = 2)	524,5005 ms
Fixed-base windowing method (window width = 3)	429,6207 ms
Fixed-base windowing method (window width = 4)	395,1432 ms
Fixed-base windowing method (window width = 5)	452,7785 ms
Fixed-base windowing method (window width = 6)	566,9521 ms
Window NAF method for point multiplication (window width = 2)	1189,8801 ms
Window NAF method for point multiplication (window width = 3)	1030,2555 ms
Window NAF method for point multiplication (window width = 4)	980,302 ms
Window NAF method for point multiplication (window width = 5)	946,4458 ms
Window NAF method for point multiplication (window width = 6)	908,9106 ms

De la tabla anterior, podemos observar que el algoritmo que más rápido realizó las 100 multiplicaciones es el "Fixed-base windowing method" con un ancho de ventana igual a 4. A partir de estos resultados, elegimos este algoritmo para la implementación de la Multiplicación de un Punto por un Escalar de la Sub-Librería de Operaciones Aritméticas. Las implementaciones en C# de estos algoritmos puede observarse en el apéndice B.2.1.5.

4.3.4. Servidor Web-API

El Servidor Web-API es el encargado de almacenar la información y responder a los llamados RESTful efectuadas desde la Aplicación Cliente. Para detallar cómo el Servidor Web-API da persistencia a los datos del sistema, y cómo se efectúan las comunicaciones con cada usuario mediante la Aplicación Cliente, nos valdremos de un Diagrama Entidad-Relación, y Diagramas de Secuencia que describen cómo se comporta el servidor ante los diferentes usos del sistema.

4.3.4.1. Diagrama Entidad-Relación

La base de datos del sistema poseerá la estructura detallada en el Diagrama Entidad-Relación de la figura 4.5 . Las entidades almacenadas son las siguientes:

Usuarios Esta entidad alberga los usuarios del sistema, con sus credenciales de acceso, fecha de alta, fecha de modificación de credenciales y su rol dentro del sistema.

Roles En esta entidad se almacenan los diferentes roles existentes en el sistema.

Claves En la entidad "Claves" se almacenan las claves privadas encriptadas de forma segura de cada usuario, junto con los datos del certificado que tiene asignado.

Firmas Finalmente, la entidad "Firmas" almacena un registro de todas las firmas electrónicas que fueron generadas mediante el sistema.

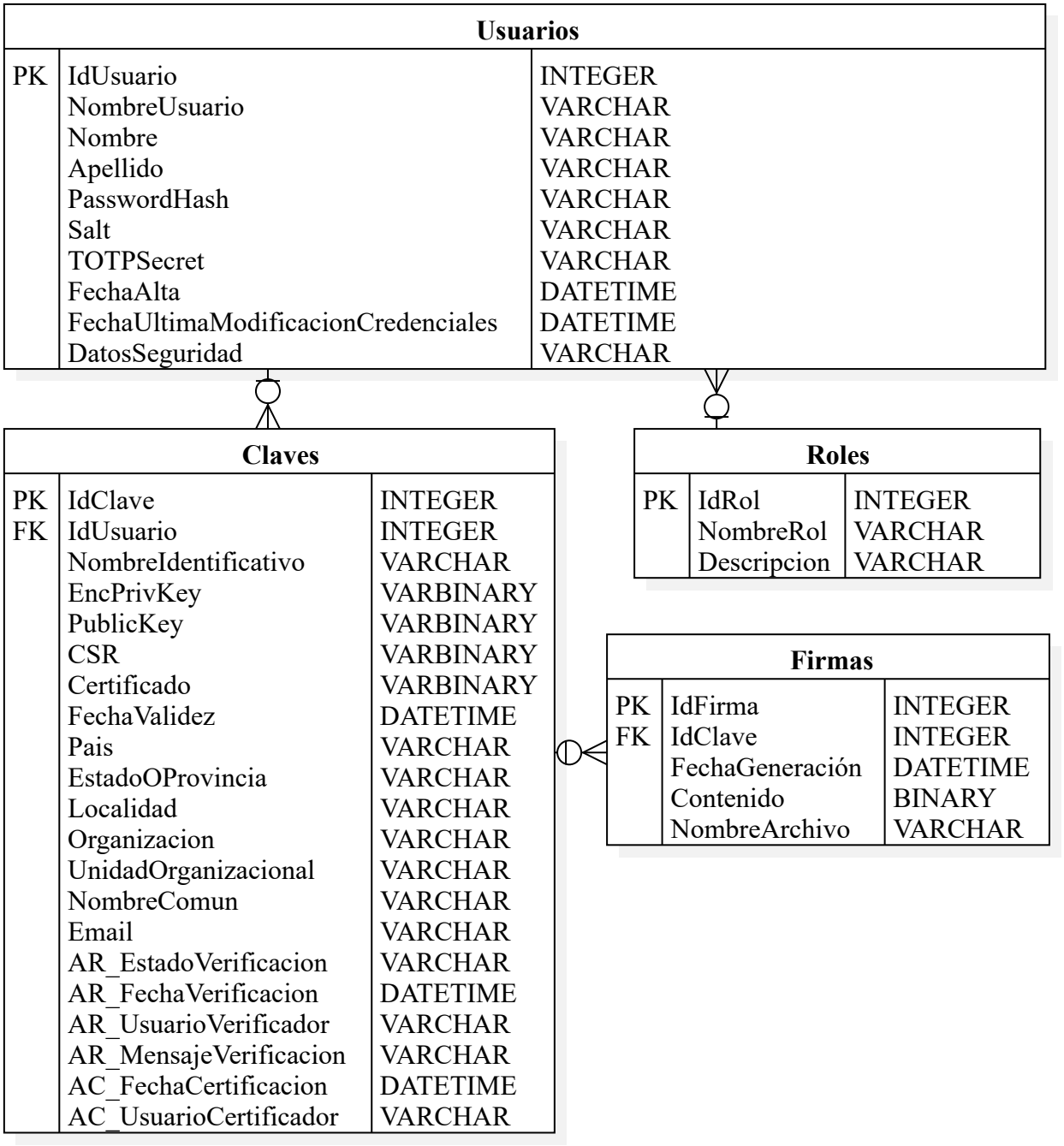

Figura 4.5: Diagrama Entidad-Relación

4.3.4.2. Diagramas de secuencia

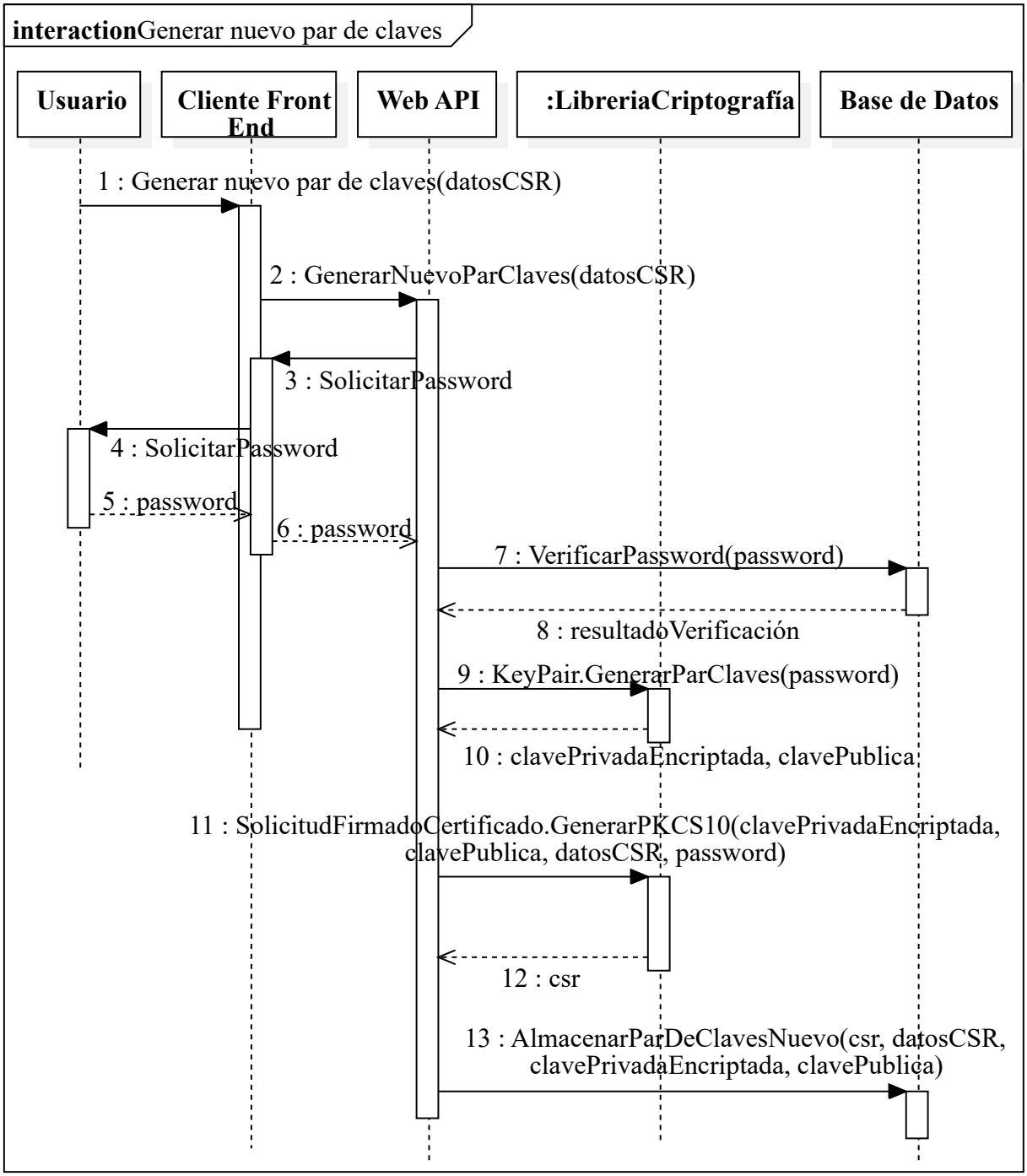

Figura 4.6: Diagrama de Secuencia: Generar nuevo par de claves y Solicitud de Firmado de Certificado

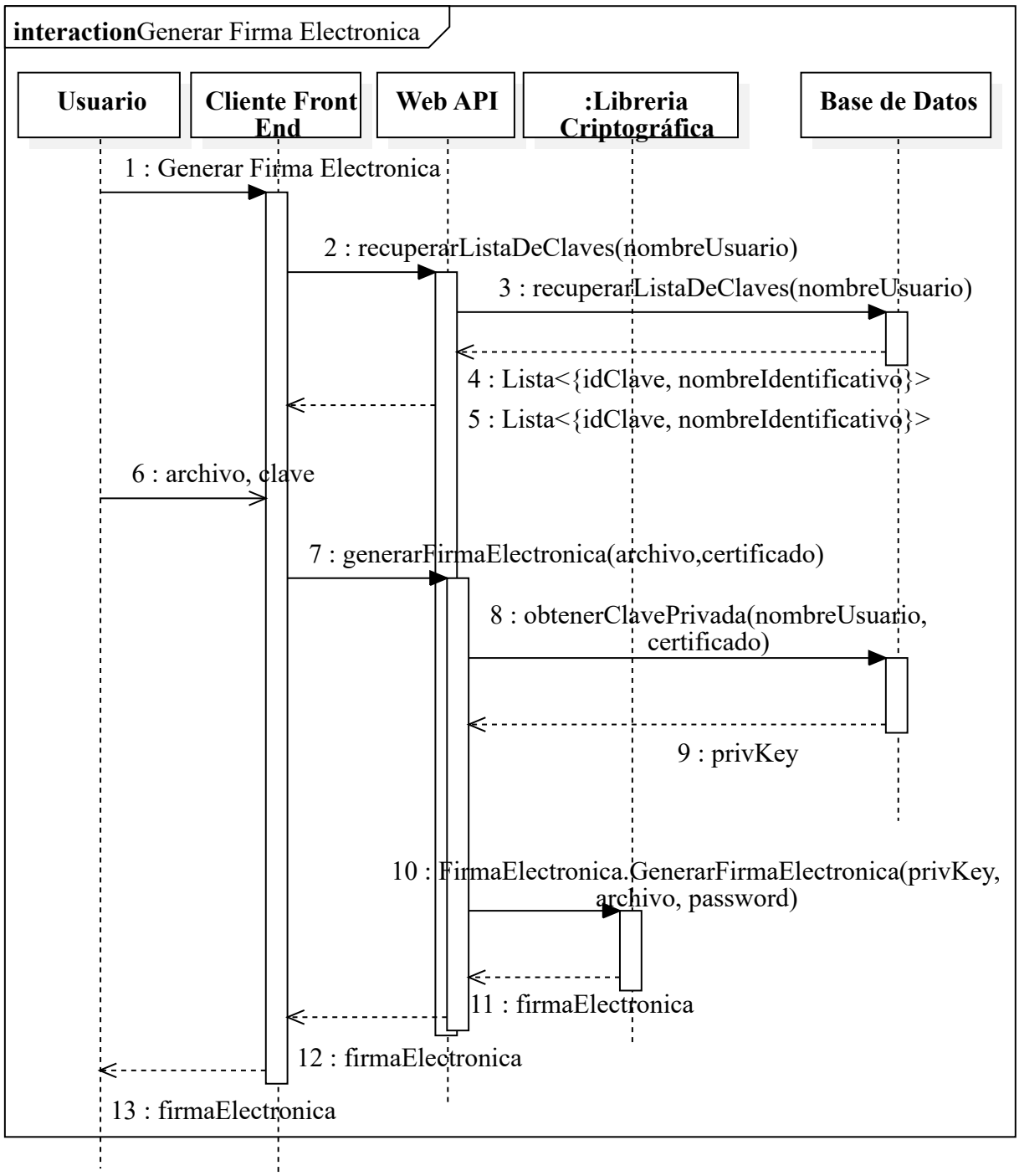

Figura 4.7: Diagrama de Secuencia: Firmar Archivo Digitalmente

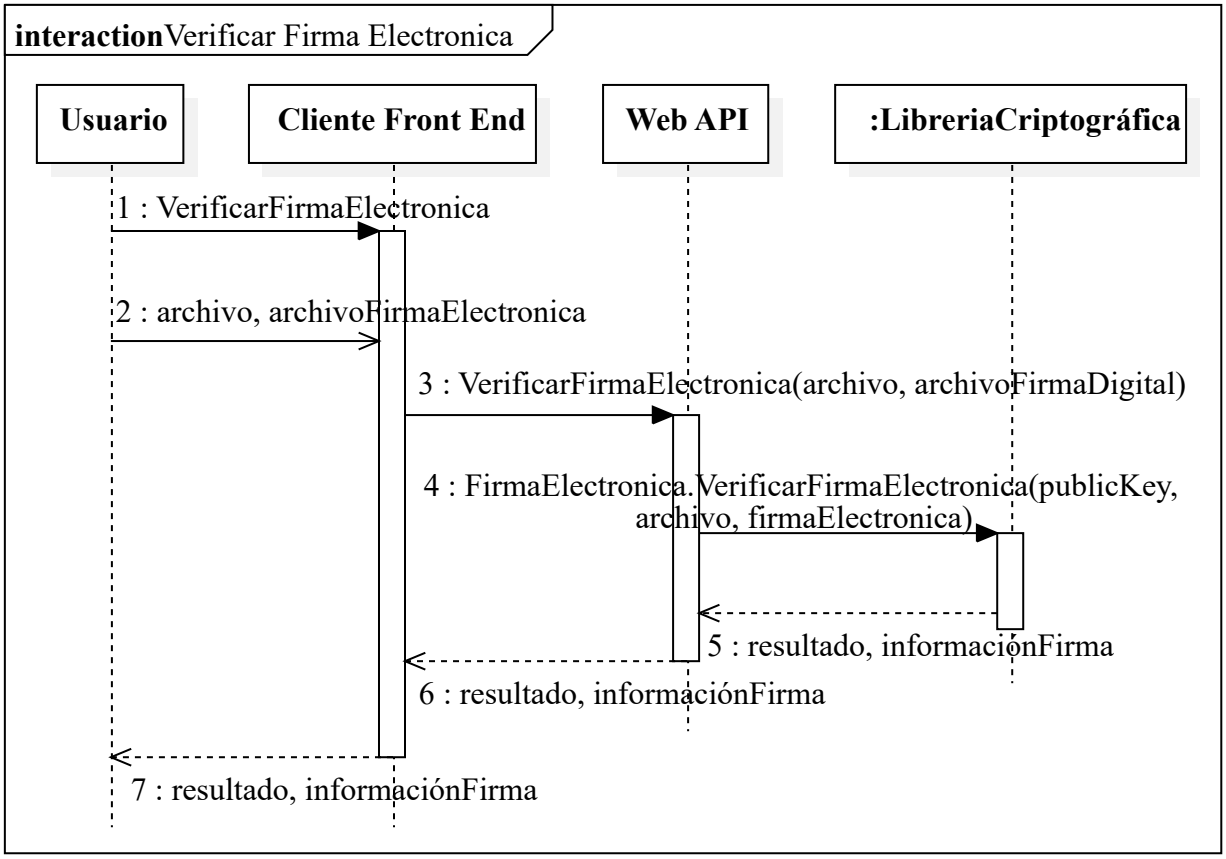

Figura 4.8: Diagrama de Secuencia: Verificar Firma Electrónica

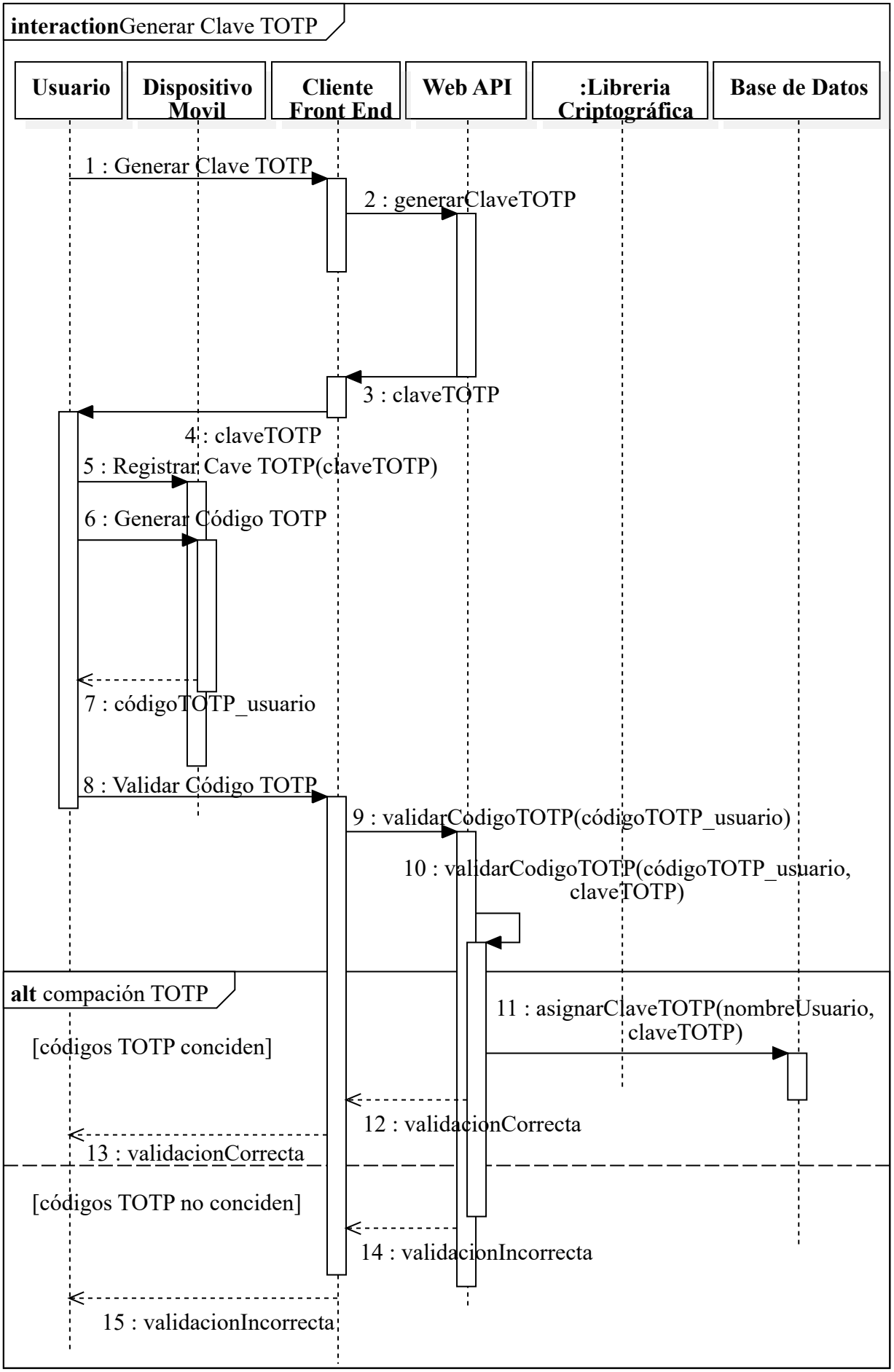

Figura 4.9: Diagrama de Secuencia: Generar Clave TOTP

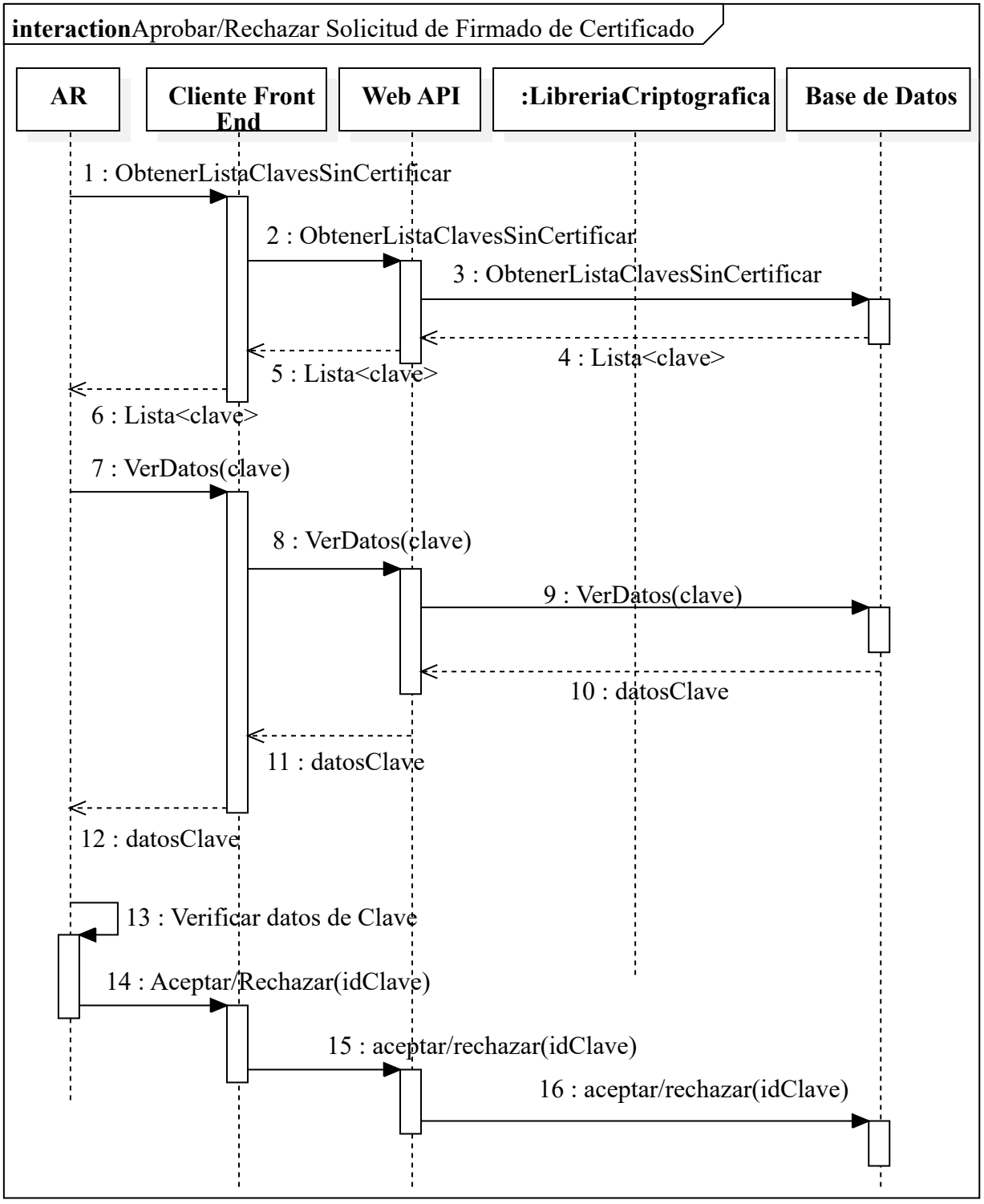

Figura 4.10: Diagrama de Secuencia: Aprobar o rechazar Solicitud de Firmado de Certificado CSR)

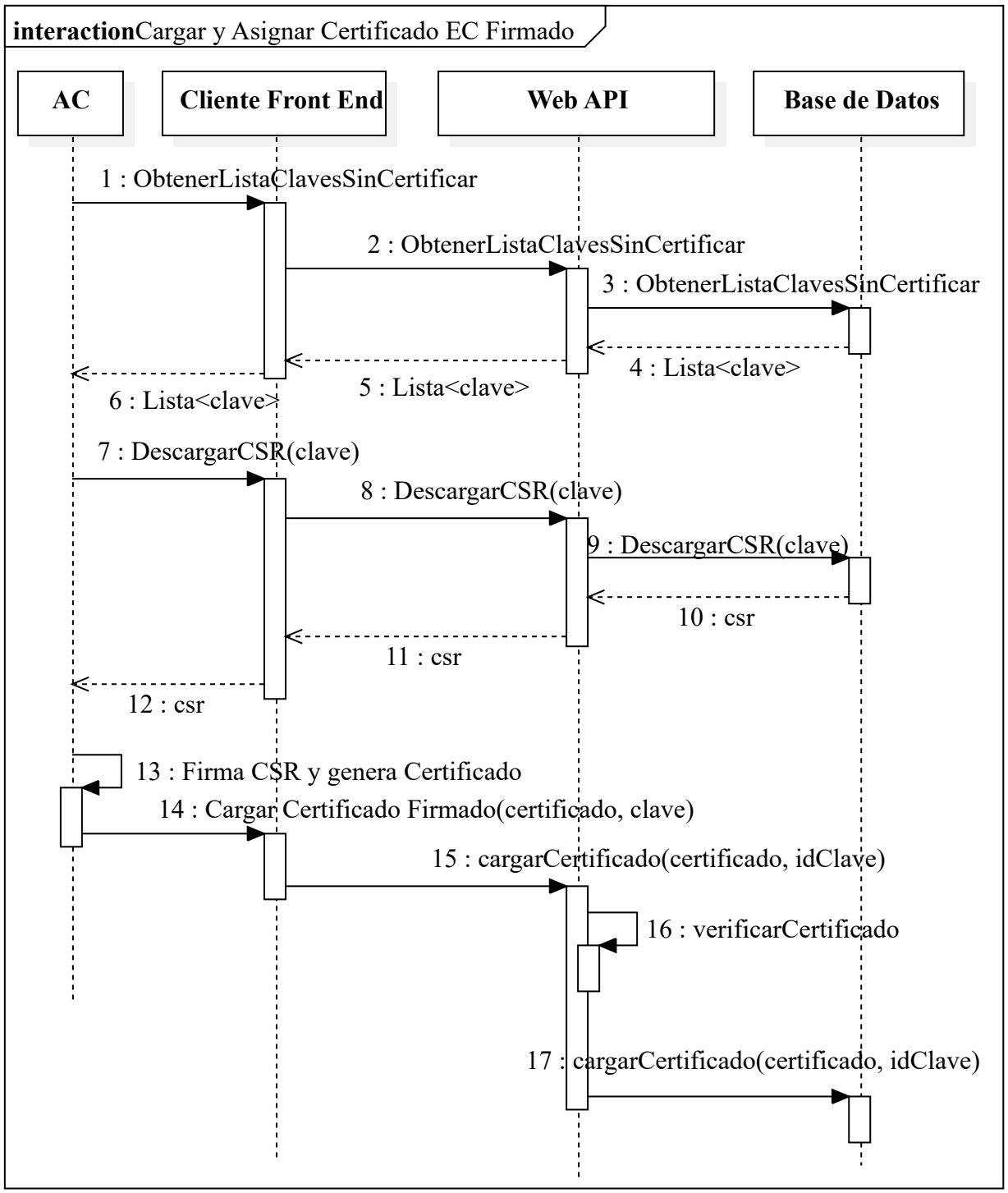

Figura 4.11: Diagrama de Secuencia: Cargar y Asignar Certificado EC Firmado

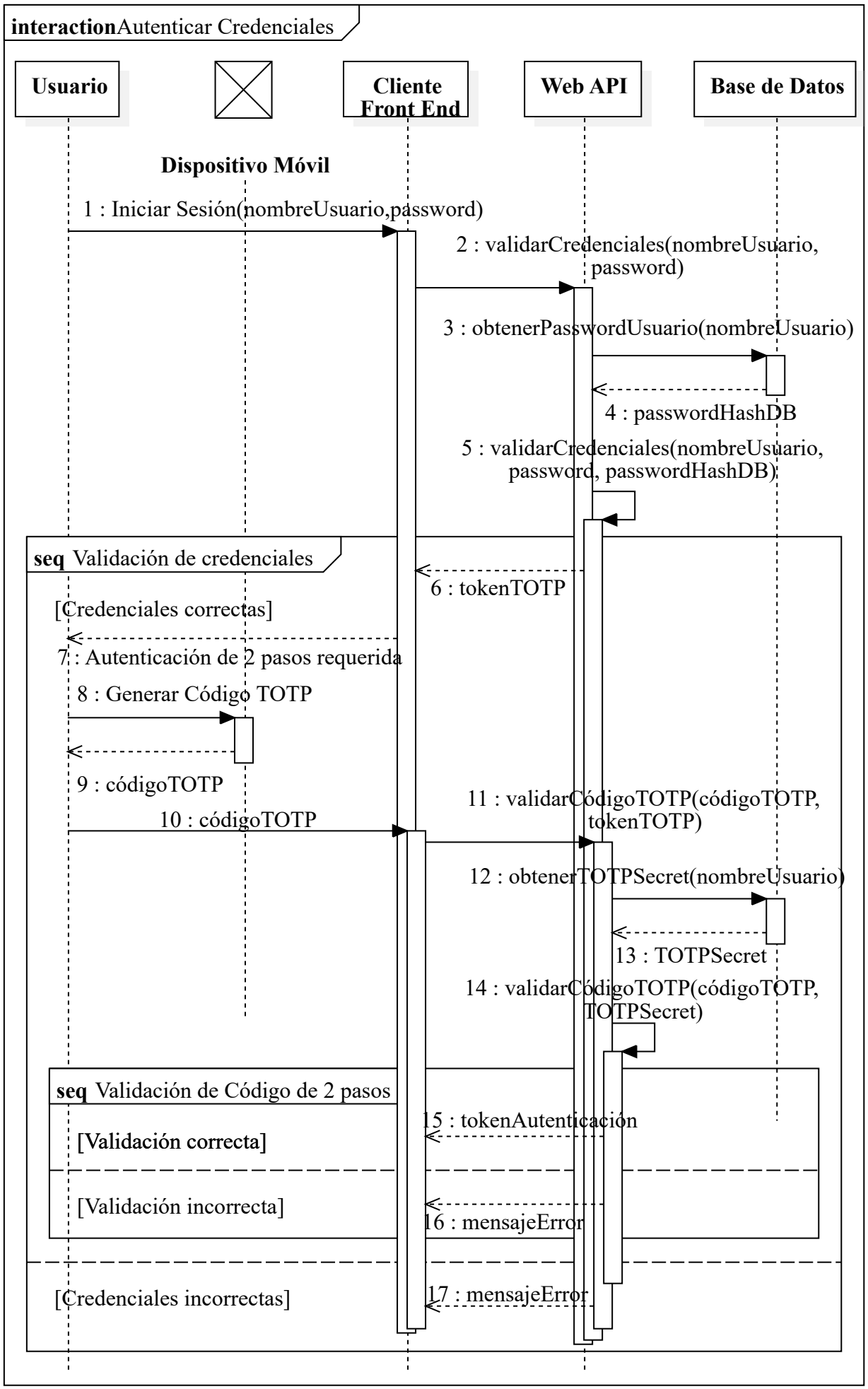

Figura 4.12: Diagrama de Secuencia: Autenticar Credenciales

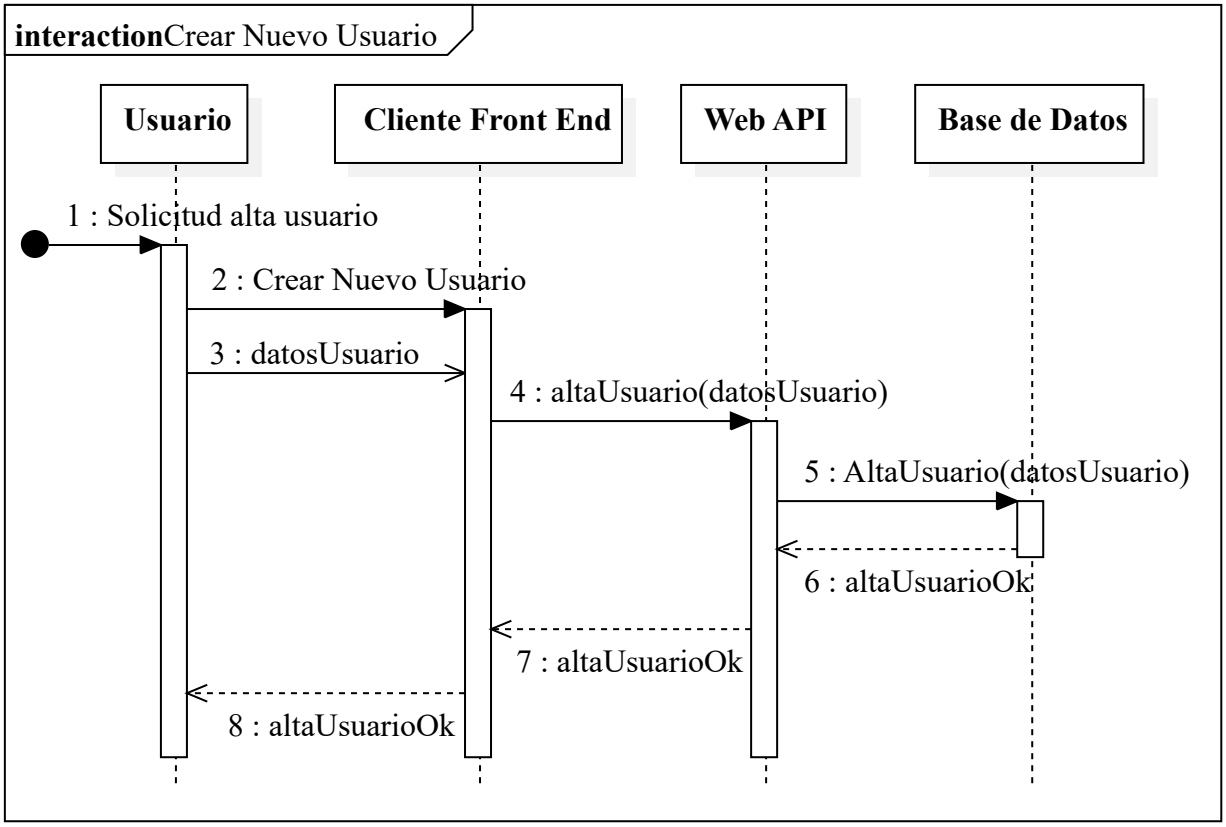

Figura 4.13

4.3.5. Aplicación Cliente

4.3.5.1. Modelo de Navegación

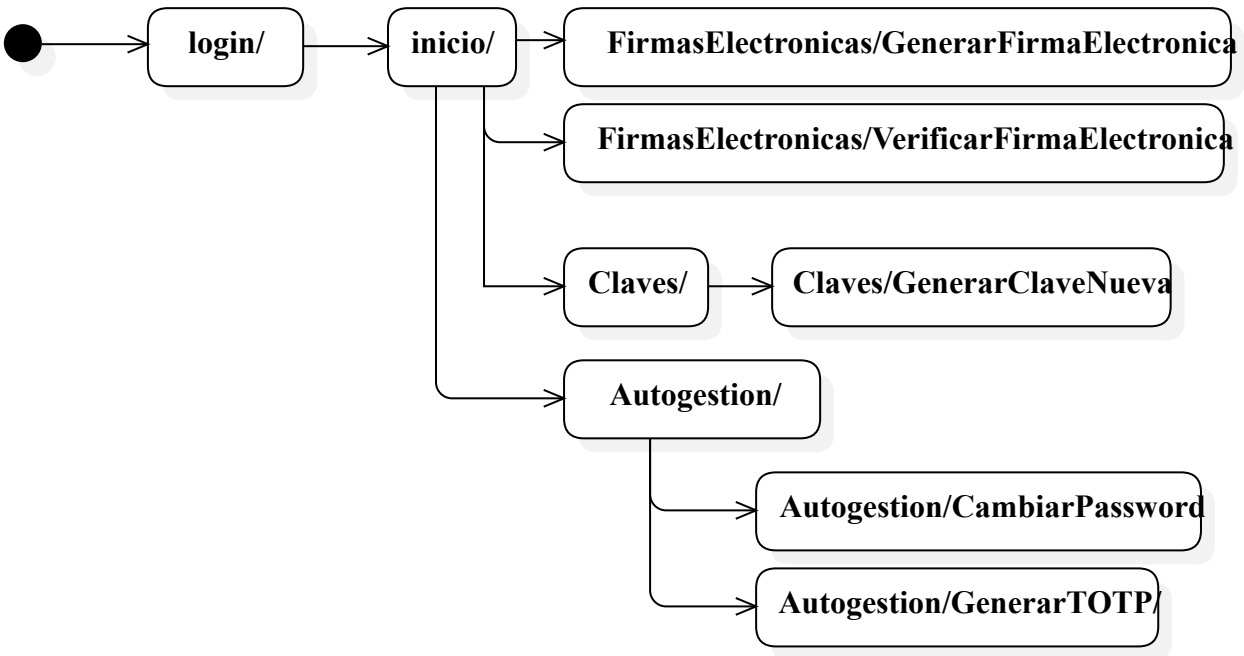

Figura 4.14: Modelo de Navegación de Usuario Normal

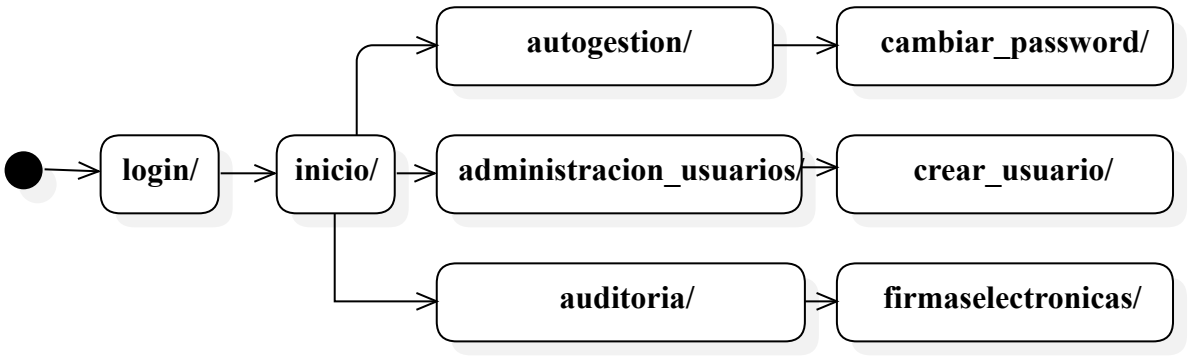

Figura 4.15: Modelo de Navegación de Administrador

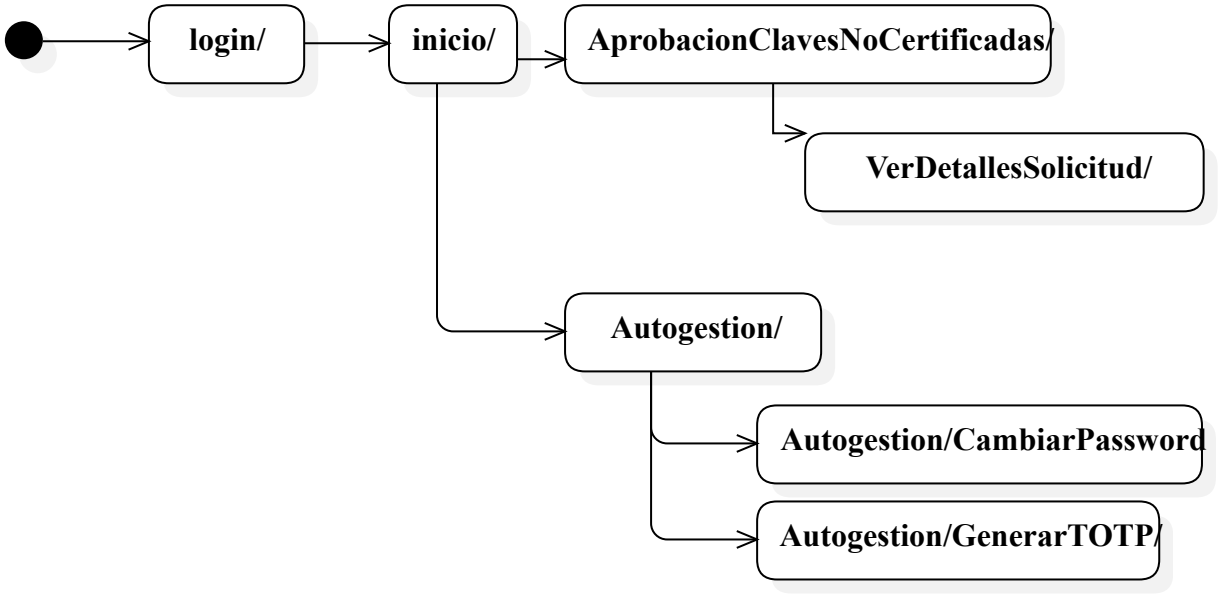

Figura 4.16: Modelo de Navegación de Autoridad de Registro

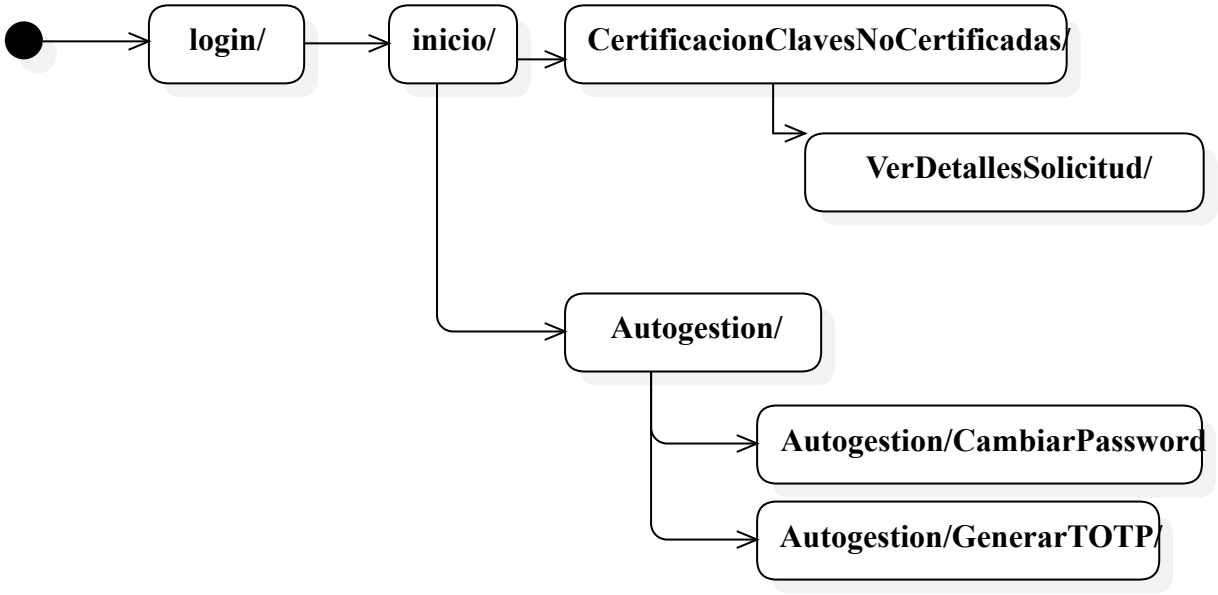

Figura 4.17: Modelo de Navegación de Autoridad Certificante

4.3.5.2. Modelo de presentación

Ingreso al sistema El ingreso al sistema se realizará empleando un formulario de Nombre de Usuario y Contraseña.

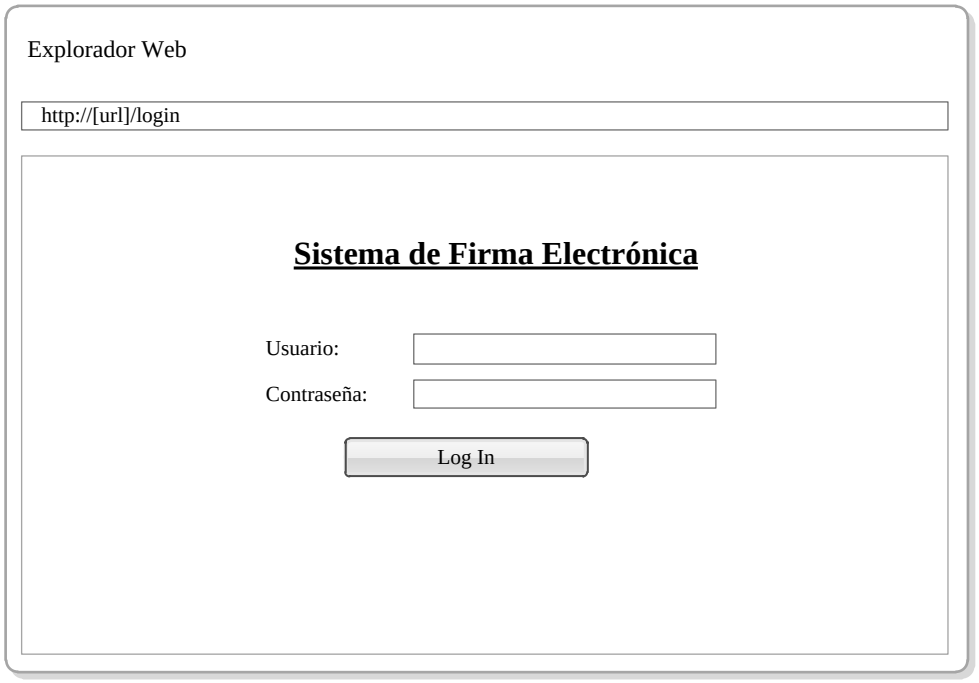

Figura 4.18: Modelo de presentación: /login (Ingreso de Credenciales)

De ser correctas las credenciales de ingreso, se solicitará la clave temporal, generada mediante el dispositivo móvil del usuario.

Figura 4.19: Modelo de presentación: /login (Ingreso de Clave Temporal)

Modelos de presentación del Usuario Normal Los usuarios normales, al ingresar al sistema, pueden ver su lista de claves creadas, clasificadas según su estado de certificación.

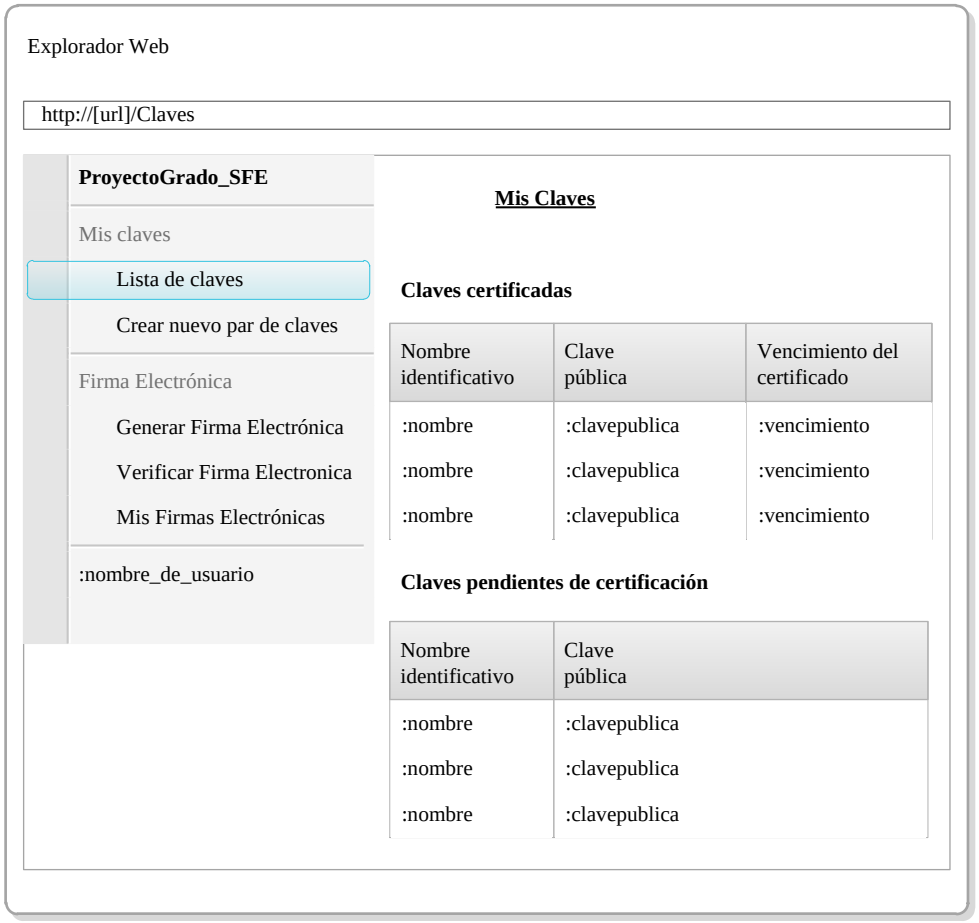

Figura 4.20: Modelo de presentación: /Claves (Lista de claves)

Seleccionando una clave, ingresan a ver los detalles de la misma, con la información suministrada por la Autoridad de Registro y la Autoridad Certificante.

Figura 4.21: Modelo de presentación: Detalles de clave

Al crear un nuevo par de claves, deben ingresar un nombre identificativo y completar los campos para la creación de la Solicitud de Firmado de Certificado, según la especificación PKCS#10.

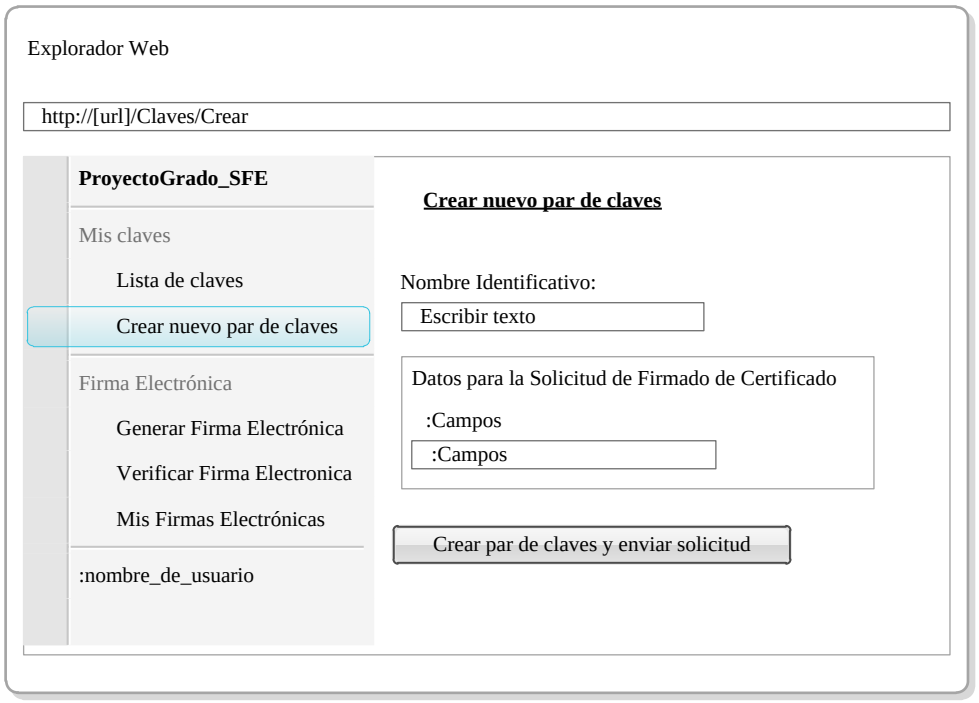

Figura 4.22: Modelo de presentación: Crear par de claves y Solicitud de Firmado de Certificado

La generación de Firmas Electrónicas se realizan mediante un formulario de selección de clave y archivo a firmar.

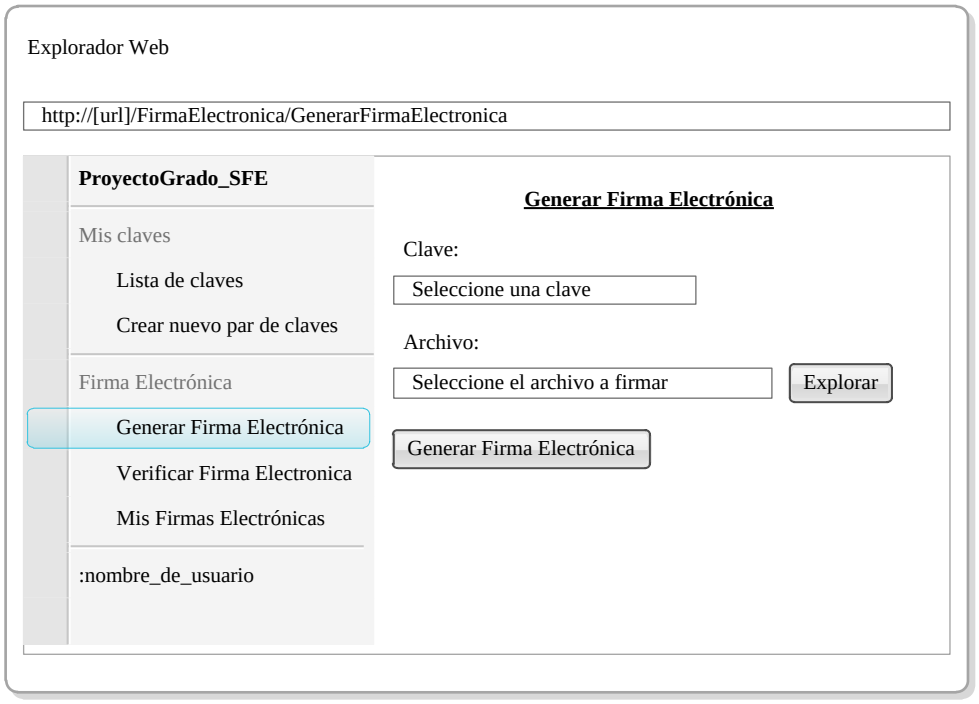

Figura 4.23: Modelo de presentación: Generación de Firma Electrónica

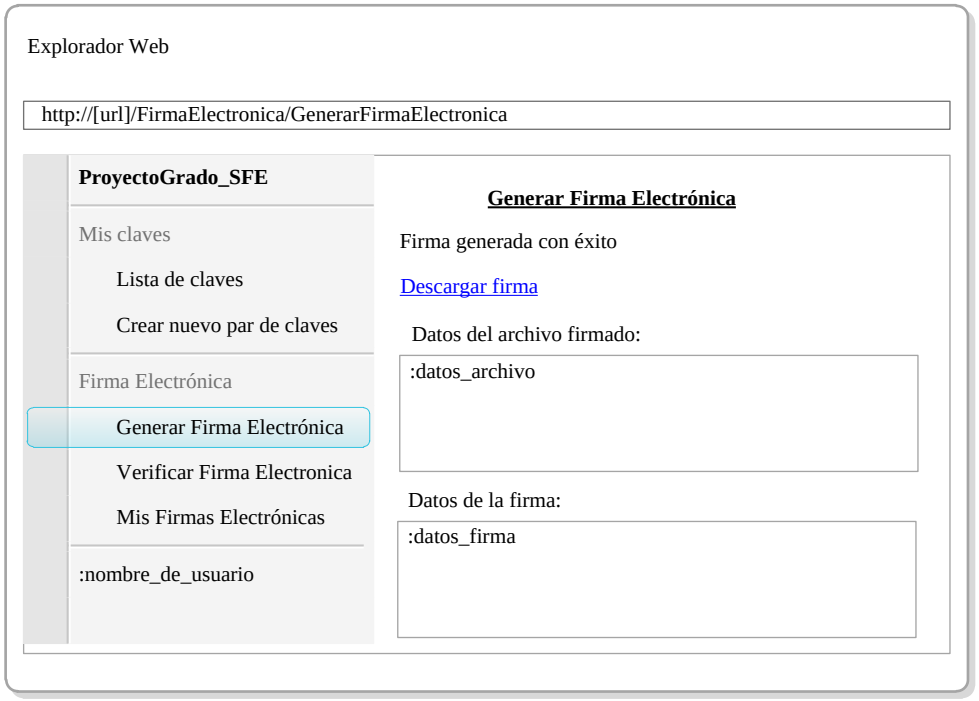

Figura 4.24: Modelo de presentación: Detalles y descarga de Firma Electrónica generada

Para realizar la verificación de un archivo junto con su firma electrónica, en la opción de menú “Verificar Firma Electrónica” se deberá cargar ambos archivos, para ver su resultado.

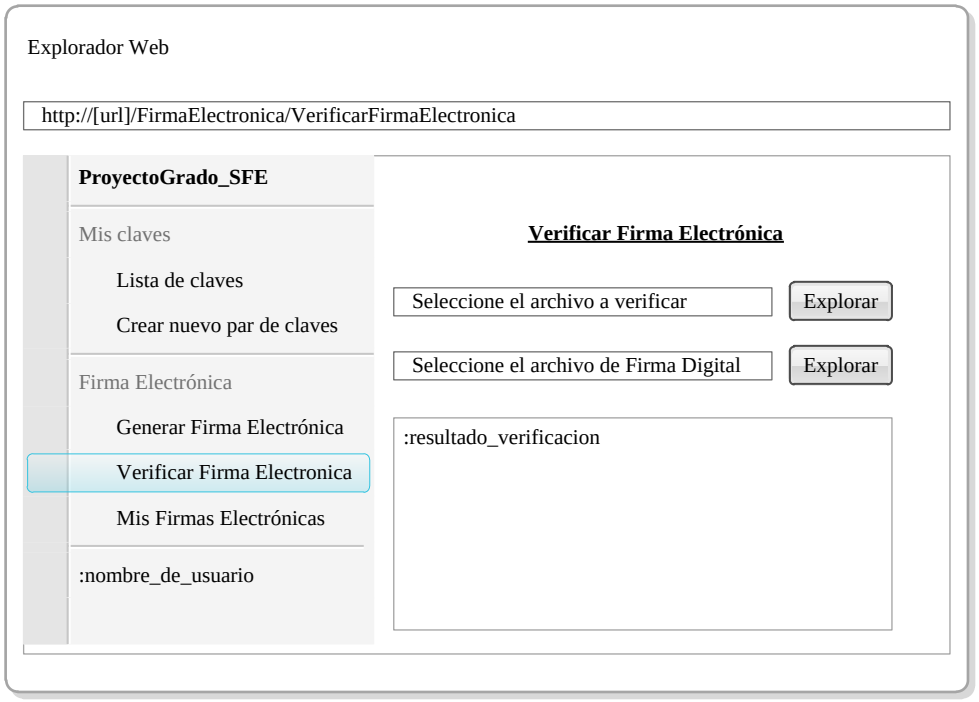

Figura 4.25: Modelo de presentación: Verificación de Firma Electrónica

Modelos de presentación de la Autoridad de Registro Al ingresar al sistema, los usuarios de la Autoridad de Registro pueden ver una lista de claves pendientes de verificación.

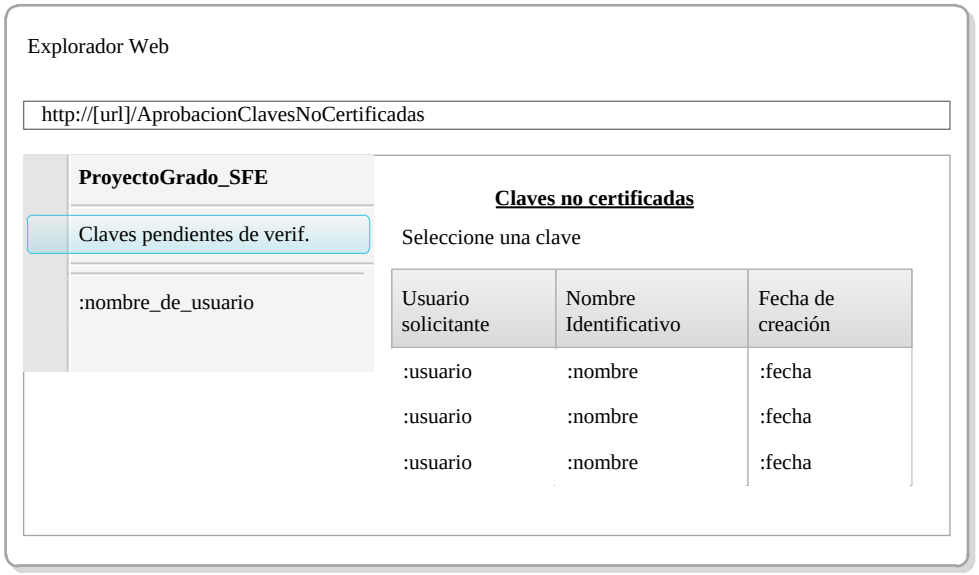

Figura 4.26: Modelo de presentación: Lista de claves no certificadas para aprobación de Autoridad de Registro

Seleccionando una de las claves de la lista, ingresan a ver los detalles de la solicitud, a partir de los cuales se decide si la solicitud es aceptada o rechazada.

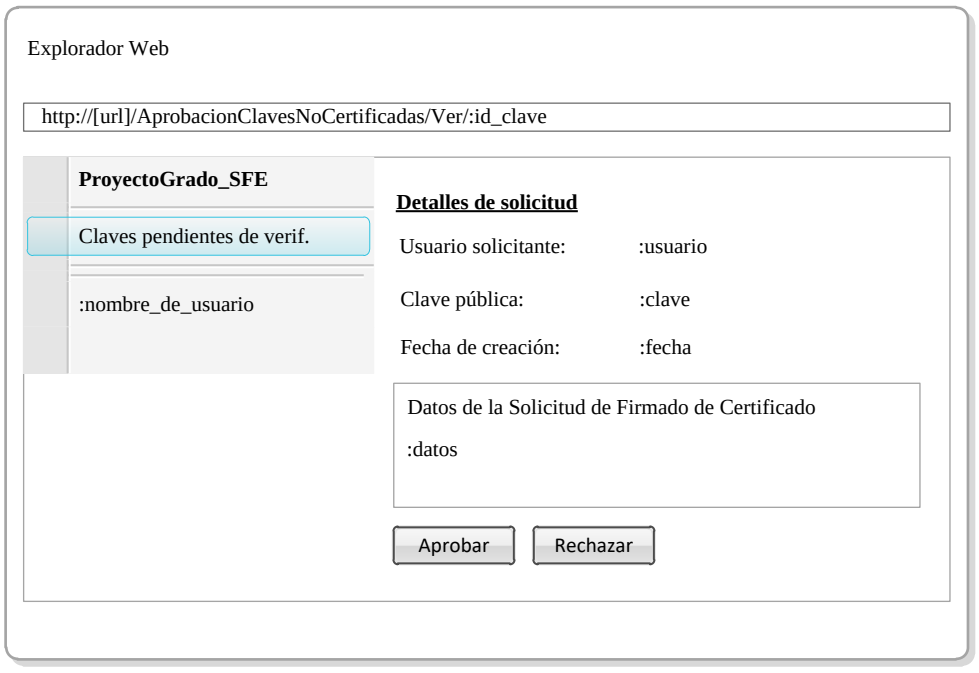

Figura 4.27: Modelo de presentación: Aprobación de Solicitud de Firmado de Certificado

Modelos de presentación de la Autoridad Certificante Los usuarios de la Autoridad Certificante, al ingresar, pueden observar una lista de claves que fueron aprobadas por la Autoridad de Registro y que están pendientes de certificación.

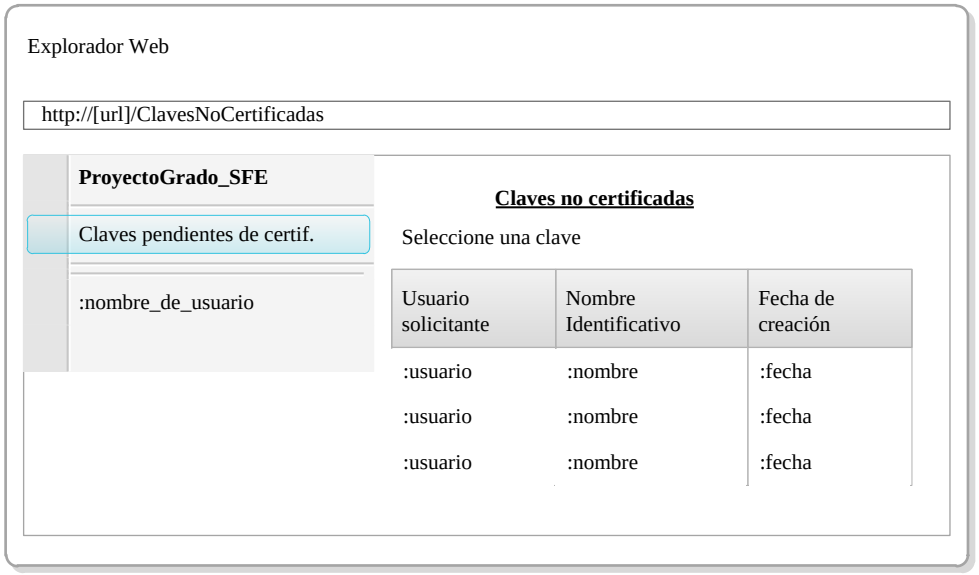

Figura 4.28: Modelo de presentación: Lista de claves no certificadas para certificación (Autoridad Certificante)

Seleccionando una clave, ingresan a ver los detalles de la misma. Deberán descargar la Solicitud de Firmado de Certificado (un archivo de tipo ***.csr*** que contiene la solicitud en formato PKCS#10) para firmarla, generando el certificado, y poder subirlo en la misma página.

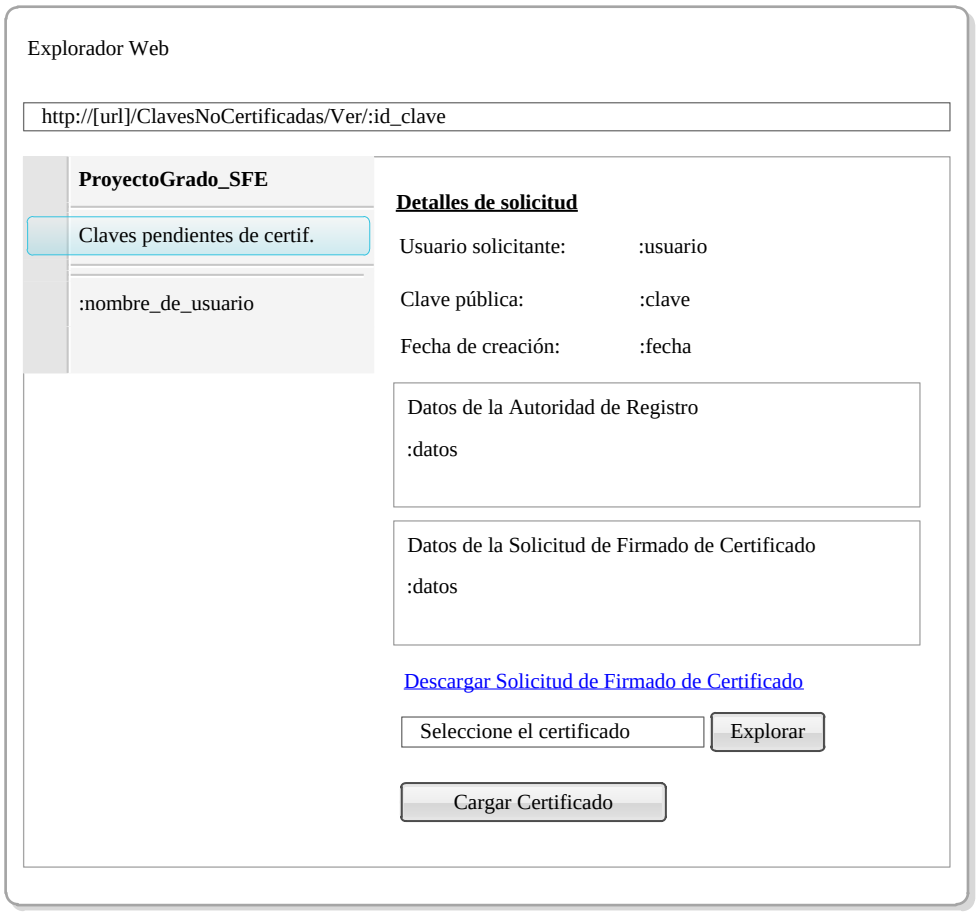

Figura 4.29: Modelo de presentación: Detalles de Solicitud de Firmado de Certificado (Autoridad Certificante)

Modelos de presentación del Administrador del Sistema Los usuarios Administradores del Sistema pueden crear usuarios y ver un listado de los que actualmente se encuentran en el sistema:

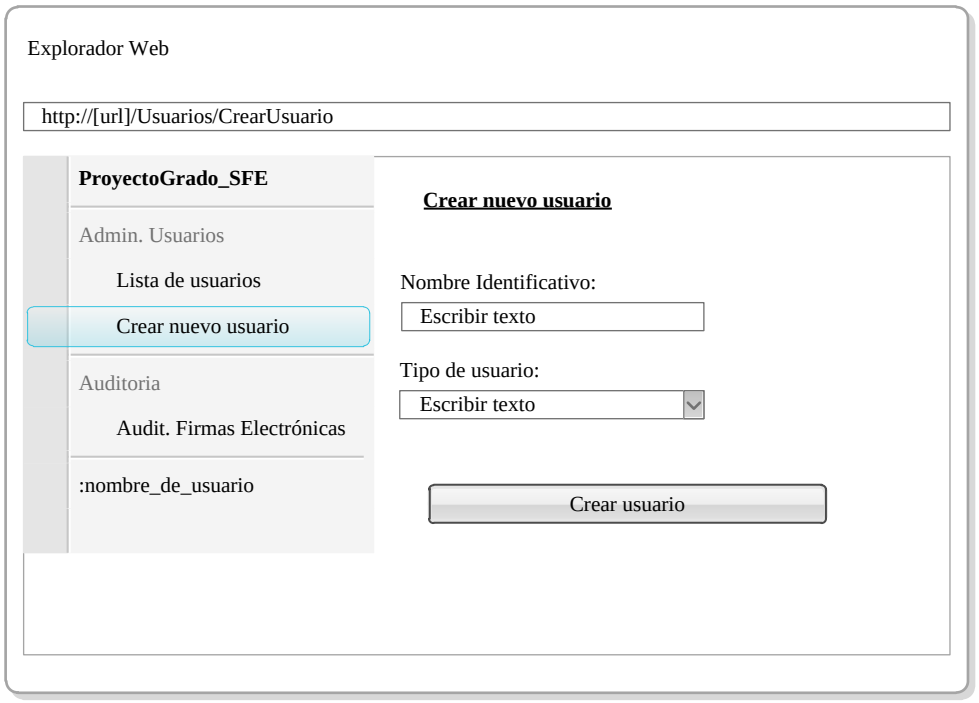

Figura 4.30: Modelo de presentación: Creación de nuevo usuario

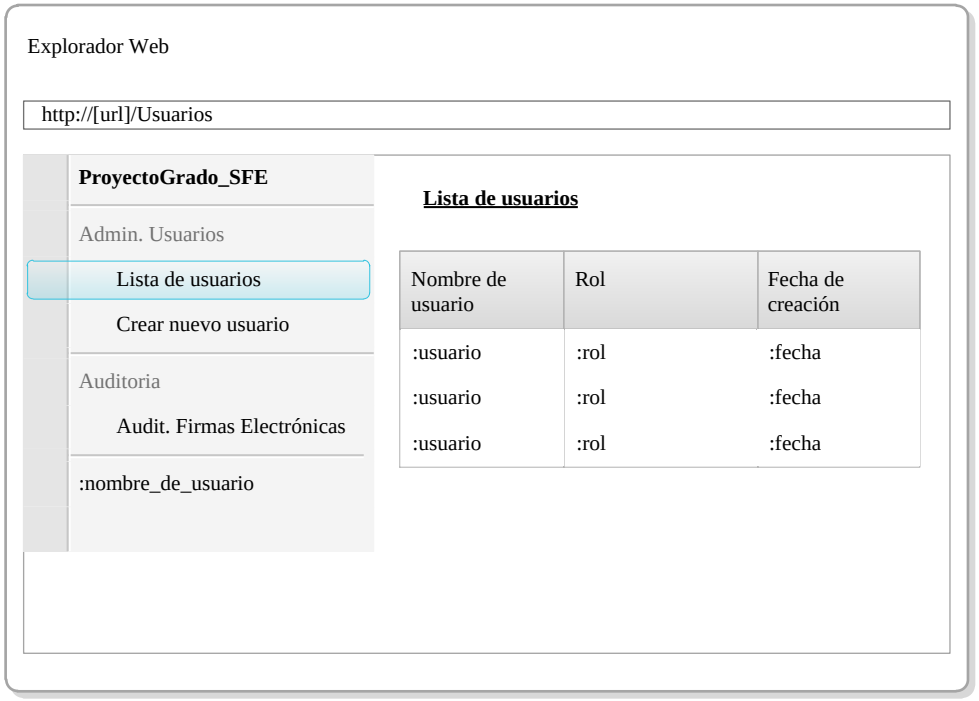

Figura 4.31: Modelo de presentación: Listado de usuarios registrados

A su vez, los usuarios Administradores del Sistema pueden auditar las Firmas Electrónicas que fueron generadas empleando el sistema:

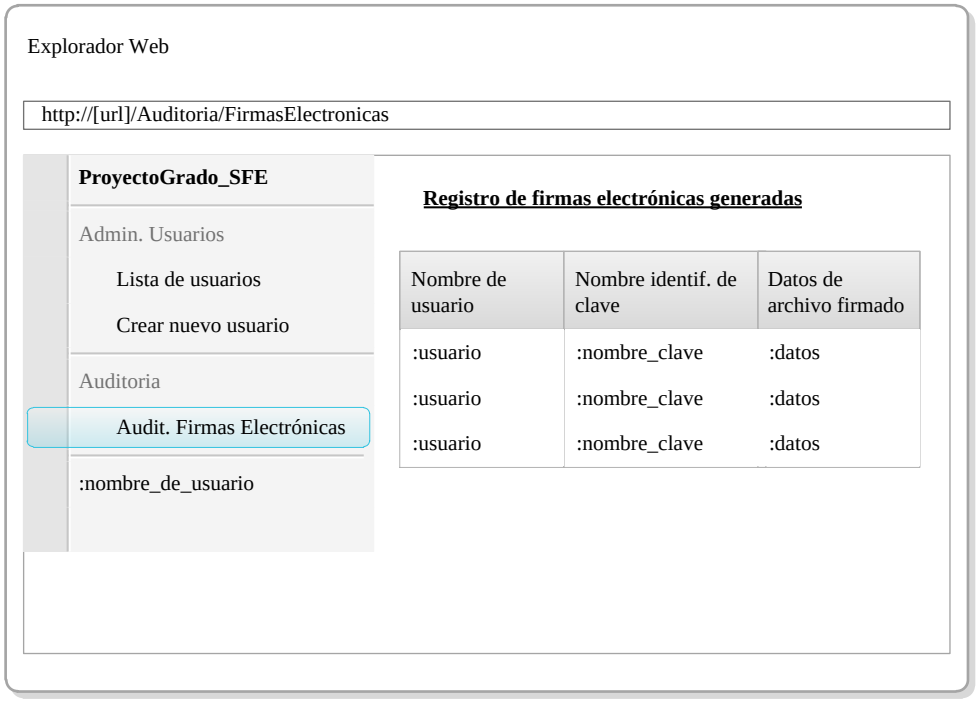

Figura 4.32: Modelo de presentación: Auditoría de Firmas Electrónicas

Administración de cuenta Todos los tipos de usuario pueden acceder a la página de Administración de Cuenta seleccionando su nombre de usuario en el menú principal. En la misma, se presentan opciones para cambio de contraseña o regeneración del código temporal.

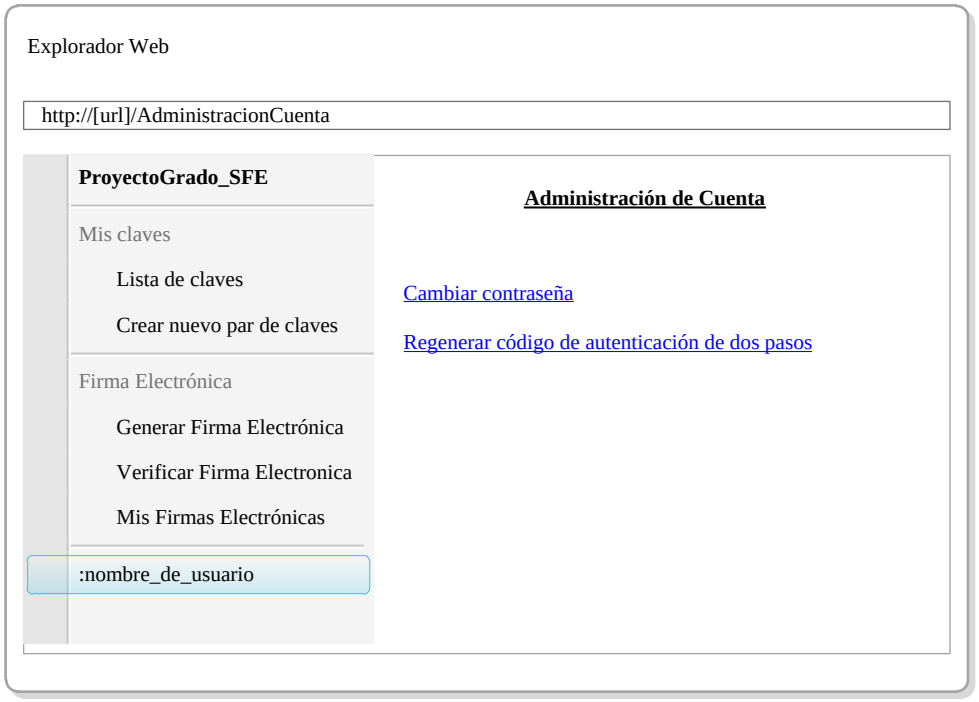

Figura 4.33: Modelo de presentación: Administración de Cuenta

Figura 4.34: Modelo de presentación: Regeneración de código temporal

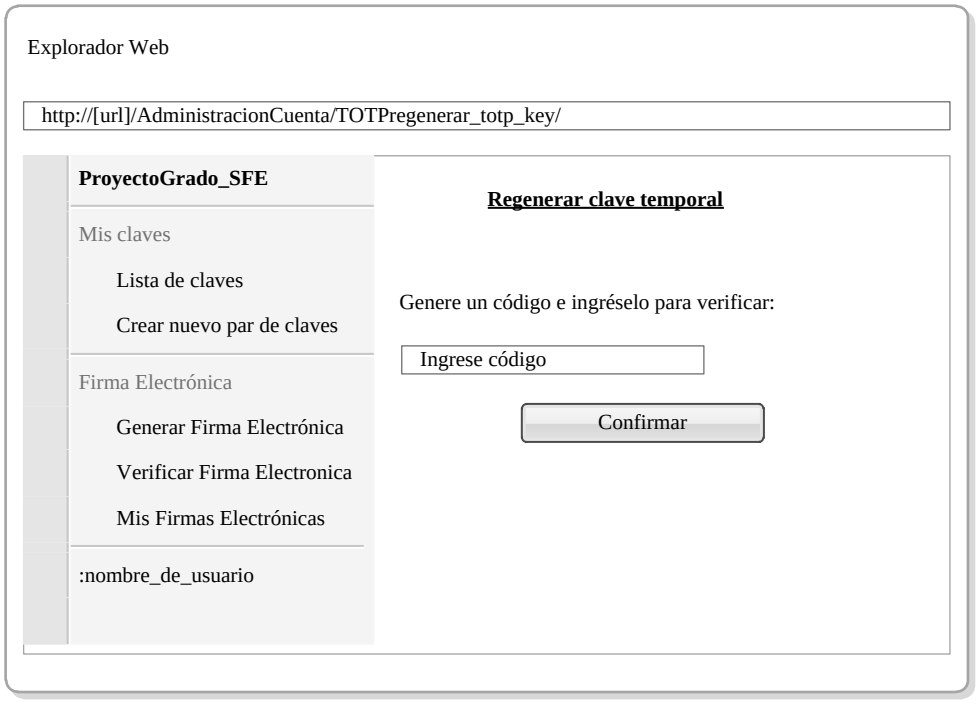

Figura 4.35: Modelo de presentación: Confirmación de código temporal

Figura 4.36: Modelo de presentación: Mensaje de confirmación de código temporal

4.3.6. Información adicional sobre la implementación del Sistema de Firma Electrónica

En este apartado describiremos diferentes cuestiones a tener en cuenta para la correcta implementación del presente proyecto.

Es de vital importancia prestar atención a todas las cuestiones relacionadas a seguridad informática en un sistema que va a centralizar todas las operaciones criptográficas. Por esta cuestión, debemos realizar las siguientes especificaciones:

4.3.6.1. Usuarios del sistema

El Sistema de Firma Electrónica cuenta con 4 roles (tipos de usuario), según su función en la utilización del sistema:

- Usuario Administrador: Tipo de usuario con permisos para la creación y eliminación de todos los tipos de usuario del sistema. Supervisa el buen uso y funcionamiento del mismo.
- Usuario “Autoridad de Registro”: Usuario encargado de verificar la autenticidad de los datos ingresados en las Solicitudes de Firmado de Certificado (CSR). Cuando un Usuario de Autoridad de Registro aprueba una solicitud, ésta pasa a los Usuarios de Autoridad de Registro para que generen el certificado correspondiente y lo carguen en el sistema.
- Usuario “Autoridad Certificante”: Usuario encargado de firmar las Solicitudes de Firmado de Certificado (CSR) que hayan sido aprobadas por la Autoridad de Registro.
- Usuario Normal: Usuario genérico del sistema, capaz de generar nuevos pares de claves y sus correspondientes Solicitudes de Firmado de Certificado, generar firmas electrónicas y verificar las mismas.

El Diagrama de Actividad de la figura 4.37 describe el procedimiento a seguir cuando los miembros de la organización deseen que el Usuario Administrador les brinde una cuenta de usuario del Sistema de Firma Electrónica.

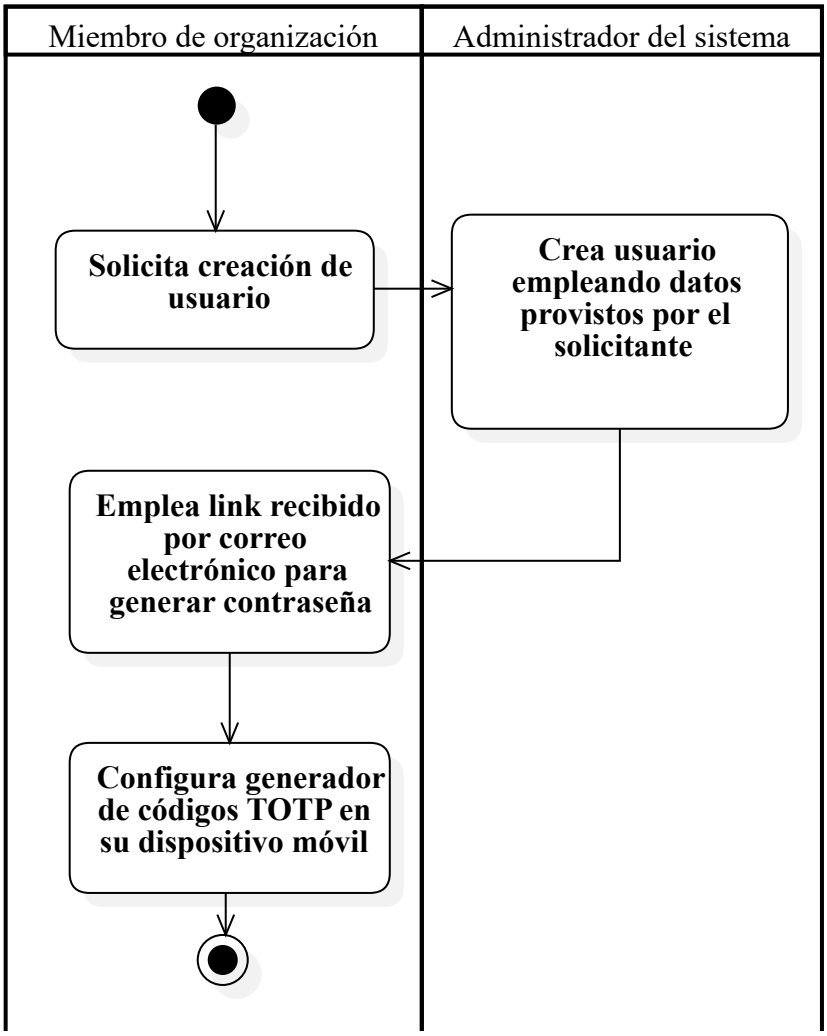

Figura 4.37: Diagrama de Actividad para la solicitud de creación de usuario en el Sistema de Firma Electrónica

4.3.6.2. Almacenamiento de hashes de contraseñas de usuarios

Almacenar los valores hash las contraseñas de los usuarios en la base de datos es una práctica ampliamente utilizada, y altamente recomendada en cualquier sistema moderno, y es una práctica que será implementada en el sistema del presente proyecto.

Ninguna contraseña será almacenada en crudo en la base de datos del sistema. Solamente se almacenará un valor "hash" de la misma, de cada usuario, empleando la función PBKDF2 (Password-Based Key Derivation Function 2). Cada vez que un usuario autentique en el sistema, será comparado el valor hash de la contraseña ingresada con el hash almacenado, para determinar si ésta es correcta.

La función PBKDF2 es parte de la especificación de los estándares de criptografía pública de los laboratorios RSA (PKCS, según sus siglas en inglés)[18], y es una de las técnicas de derivación de contraseñas más ampliamente utilizada en diversos sistemas web, y con una amplia implementación en diferentes lenguajes y frameworks.

[18]B. Kaliski. 2000. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898 (Informational). Internet Engineering Task Force.

4.3.6.3. Código Time-Based One-Time-Password

Cada usuario tendrá configurado en su dispositivo móvil un generador de claves Time-Based One-Time-Password. Estas claves se basan en el tiempo y se utilizan como credencial de acceso, junto con el password. El algoritmo utilizado en el sistema para la verificación de estos códigos será HMAC-SHA256, tal como lo recomienda la Internet Engineering Task Force[19].

En la base de datos del Sistema de Firma Electrónica, la tabla "Usuarios" poseerá una columna denominada "TOTPSecret", en la cual se almacenará el *secreto* (string definido en la especificación RFC-6238 a partir del cual se generan los códigos temporales) asignado a cada usuario. Este "secreto" es el que se emplea para generar los códigos temporales y compararlos con los provistos por los usuarios al momento de iniciar sesión.

4.3.6.4. Tokens de autenticación

Al momento de autenticar en el sistema, el cliente Front-End (la aplicación que el usuario utiliza en su explorador web) recibe un Token de Autenticación, que es empleado como credencial de autenticación. Este Token de autenticación tendrá una duración de 5 minutos, que son renovados cada vez que el usuario realiza cualquier operación o navegación dentro del sistema. De esta forma, si transcurren más de 5 minutos de inactividad por parte del usuario, el Token expirará y el usuario deberá volver a autenticarse.

Este token estará generado empleando el estándar ***JSON Web Tokens***[20], cuyo funcionamiento se basa en la utilización de una clave secreta almacenada en el servidor para firmar, empleando el algoritmo HMAC-SHA256, una serie de datos de autenticación. Estos datos firmados conforman el token que el servidor envía al cliente. Éste último, cada vez que genera una llamada al Servidor WebAPI, debe proveer su token asignado para poder ejecutar las operaciones.

Para incrementar la seguridad en la generación de tokens de acceso, la clave secreta que emplea el servidor será generada de forma aleatoria cada 24 horas. Además, será únicamente localizada en memoria RAM, evitando almacenarla en un archivo en disco rígido.

4.3.6.5. Almacenamiento de Claves Privadas

Para un almacenamiento seguro de las claves privadas generadas en el Sistema de Firma Electrónica, éstas serán encriptadas empleando criptografía de clave simétrica. Mediante criptografía de clave simétrica, se logra cifrar información empleando una única clave secreta.

La especificación criptográfica de clave simétrica a emplear para encriptar las claves privadas será AES-256, en modo Cipher Block Chaining (CBC), tal como lo recomiendan la *Bundesamt für Sicherheit in der Informationstechnik (Oficina federal de seguridad de la información, Alemania)*[21], *National Institute of Standards and Technology (Instituto nacional de estándares y tecnología, Estados Unidos)*[22], y *Agence nationale de la sécurité des systèmes d'information (Agencia nacional de seguridad de sistemas de información, Francia)*[23].

AES-256 es una especificación criptográfica de clave simétrica (se emplea una sola clave secreta para encriptar y desencriptar), que emplea una clave secreta de 256 bits, y un *Vector de Inicialización (IV)* de 128 bits. Este último es una secuencia aleatoria de 128 bits, único por cada operación de encriptación, que puede ser de conocimiento público y que se emplea para proporcionar aleatoriedad a los criptogramas generados. De esta forma, no se generan nunca cifrados repetidos, empleando la misma clave secreta y texto plano.

[19] David M'Raihi et al. 2011. *RFC 6238-TOTP: Time-based one-time password algorithm*.
[20] M. Jones et al. 2015. *RFC 7519: JSON Web Token (JWT)*.
[21] Bundesamt für Sicherheit in der Informationstechnik 2017.
[22] NIST (National Institute of Standards and Technology) 2016.
[23] Agence nationale de la sécurité des systèmes d'information 2014.

Cada usuario propietario de pares de claves públicas y privadas en el Sistema de Firma Electrónica, poseerá además una única clave secreta de 256 bits (denominada K1) generada de forma aleatoria y segura en el servidor, empleada para encriptar ***todas*** sus claves privadas. Cada vez que el usuario cree un nuevo par de claves para emplear en el Sistema de Firma Electrónica, la clave privada del par será encriptada utilizando K1 y un vector de inicialización generado de forma aleatoria. Cuando el usuario desee emplear esta clave privada para generar una firma electrónica, ésta deberá ser desencriptada empleando la clave secreta K1 del usuario almacenada en la Base de Datos.

Es necesario que esta clave K1 no se encuentre almacenada como texto plano en la Base de Datos, debido a que un intruso que logre acceder a las tablas de la Base de Datos, podrá ver las claves K1 y las claves privadas encriptadas de todos los usuarios, accediendo a información crítica del Sistema de Firma Electrónica. Es por esta razón que las claves K1 de cada usuario serán encriptadas empleando la misma especificación criptográfica AES-256. Esta "secunda etapa" de encriptación se realizará utilizando la contraseña de acceso al sistema de *cada* usuario.

Debido a que las contraseñas de los usuarios son secuencias de 6 a 16 caracteres alfanuméricos y que el método criptográfico AES-256 utiliza claves de 256 bits, es necesario que la contraseña sufra una *derivación* para poder ser empleada como clave secreta AES-256. Se empleará la función de derivación de claves PBKDF2, con la cual se genera una secuencia de 256 bits única a partir de cualquier clave alfanumérica que posea el usuario.

Para comprender mejor el funcionamiento de este diseño de seguridad, los algoritmos 4.3,

4.4 y 4.5 detallan el funcionamiento del mismo.

Algoritmo 4.3: Configuración inicial de los Datos de Seguridad de un usuario

Entrada: Contraseña del usuario

Salida: Datos de Seguridad del usuario [K1 encriptado, K3, salt, IV] almacenados en Base de Datos

1 Generamos una secuencia aleatoria de 256 bits denominada **K1** y una secuencia aleatoria de 128 bits denominada **IV** (vector de inicialización).;

2 Generamos un valor "salt" aleatorio de 128 bits, y usamos PBKDF2 para generar una derivación de 512 bits a partir de la contraseña del usuario y el valor "salt" generado. Esta derivación de 512 bits es dividida en dos secuencias de 256 bits denominadas **K2** y **K3**.;

3 Usamos K2 como clave secreta para encriptar mediante AES-256 la clave K1, empleando el vector de inicialización IV generado en el paso 1. ;

4 Almacenamos en base de datos (junto con los datos del usuario) la secuencia K1 encriptada en el paso 3, la secuencia K3 y el valor "salt" generados en el paso 2, y la secuencia "IV" generada en el paso 1. ;

Algoritmo 4.4: Encriptación/Desencriptación de claves privadas

Entrada: Contraseña del usuario, clave privada encriptada o como texto plano, Datos de Seguridad del usuario [K1 encriptado, K3, salt, IV] almacenados en Base de Datos

Salida: Clave privada desencriptada/encriptada

1 Usamos PBKDF2 para hacer una derivación de 512 bits de la contraseña del usuario, empleando el valor "salt" del mismo almacenado en Base de Datos. Dividimos la derivación en dos secuencias de 256 bits cada una, denominadas **K2** y **K3**. ;

2 **si** *la secuencia K3 obtenida en el paso anterior* ***NO*** *coincide con la secuencia K3 del usuario almacenada en Base de Datos* **entonces**

3 | La contraseña ingresada por el usuario es incorrecta, no se puede continuar. ;

4 **fin**

5 **sinó**

6 | Usamos la secuencia K2 generada en el paso 1, junto con el vector de inicialización **IV** del usuario almacenado en Base de Datos para desencriptar la clave secreta **K1** del usuario, almacenada en Base de Datos. ;

7 | Empleamos la clave secreta **K1** obtenida en el paso anterior para encriptar una nueva clave privada generada por el usuario, o para desencriptar cualquiera de las claves privadas que el mismo ya posea en el Sistema de Firma Electrónica. ;

8 **fin**

Algoritmo 4.5: Regeneración de Datos de Seguridad cuando el usuario cambia su contraseña

Entrada: Contraseña anterior del usuario, contraseña nueva del usuario, Datos de Seguridad del usuario [K1 encriptado, K3, salt, IV] almacenados en Base de Datos

Salida: Nuevos Datos de Seguridad del usuario [K1 encriptado, K3, salt, IV] almacenados en Base de Datos

1 Desencriptar **K1** tal como se detalla en el algoritmo anterior. ;

2 Efectuar los pasos detallados en el algoritmo 4.3 para regenerar todos los datos de seguridad, utilizando la misma clave secreta **K1**, pero empleando la nueva contraseña del usuario. ;

4.3.6.6. Equipo Servidor a emplear para el despliegue del Sistema de Firma Electrónica

Teniendo en cuenta las características del software desarrollado en el presente proyecto de grado, los frameworks empleados y una estimación del volumen de la base de datos que acompaña al Sistema de Firma Electrónica, se propone un Equipo Servidor con las siguientes características mínimas, que deberá utilizar la organización que desee implementar el sistema:

Hardware

- Procesador Intel Core i3 de Segunda Generación o superior
- 4 GB de Memoria RAM
- 100 GB de almacenamiento persistivo (SDD preferentemente)

Software

- Sistema operativo: Ubuntu Server 16.04.3 (LTS)
- .NET Core 1.1.4 runtime (LTS)
- MySQL 5.7.19 (stable)

Debemos remarcar que se eligen versiones ”LTS”(Long Term Support) del sistema operativo y del runtime requerido para el funcionamiento del Servidor WebAPI, debido a que se opta por software que posea soporte oficial por 4 años.

El motor de base de datos MySQL deberá ser instalado en su última versión estable.

4.3.7. Consideraciones de seguridad para el despliegue

4.3.7.1. PKI en la organización

La implementación de un Sistema de Firma Electrónica en una organización, implica que ésta adoptará una Infraestructura de Clave Pública (PKI, según sus siglas en inglés que significan “Public Key Infrastructure”), en la cual es imperativo identificar los diferentes componentes y procedimientos, que garantizarán la seguridad y el correcto funcionamiento del sistema.

Dentro de esta PKI a implementar, podemos identificar los siguientes componentes:

- Usuario: Todo miembro de la organización que necesite la utilización del Sistema de Firma Electrónica, para efectuar el firmado electrónico de documentos. Dueño de pares de claves (pública y privada), las cuales empleará para realizar los firmados y verificaciones.
- Autoridad de Registro - Área de Recursos Humanos: Entidad que verifica la identidad del miembro de la organización que solicita la creación de un certificado. Dentro de la organización, la Autoridad de Registración será el Área de Recursos Humanos, quien se encargará de validar la identidad del miembro de la organización, y aprobará la misma, para luego transferir la solicitud a la Autoridad de Certificación.
- Autoridad de Certificación: Entidad dueña del Çertificado Raízz del Par de Claves Raíz. Es la encargada de recibir las Solicitudes de Firmado de Certificado aprobadas por la Autoridad de Registro - Área de Recursos Humanos y, a partir de estos, crea los certificados y los carga en el Sistema de Firma Electrónica.

De esta forma, cuando un usuario crea un par de claves para emplear en el Sistema de Firma Electrónica, debe certificarlo con su Autoridad de Registro y de Certificación pertinentes. El procedimiento para la creación de certificados a partir de un par de claves creado por el usuario puede ser detallado en el diagrama de actividad de la figura 4.38.

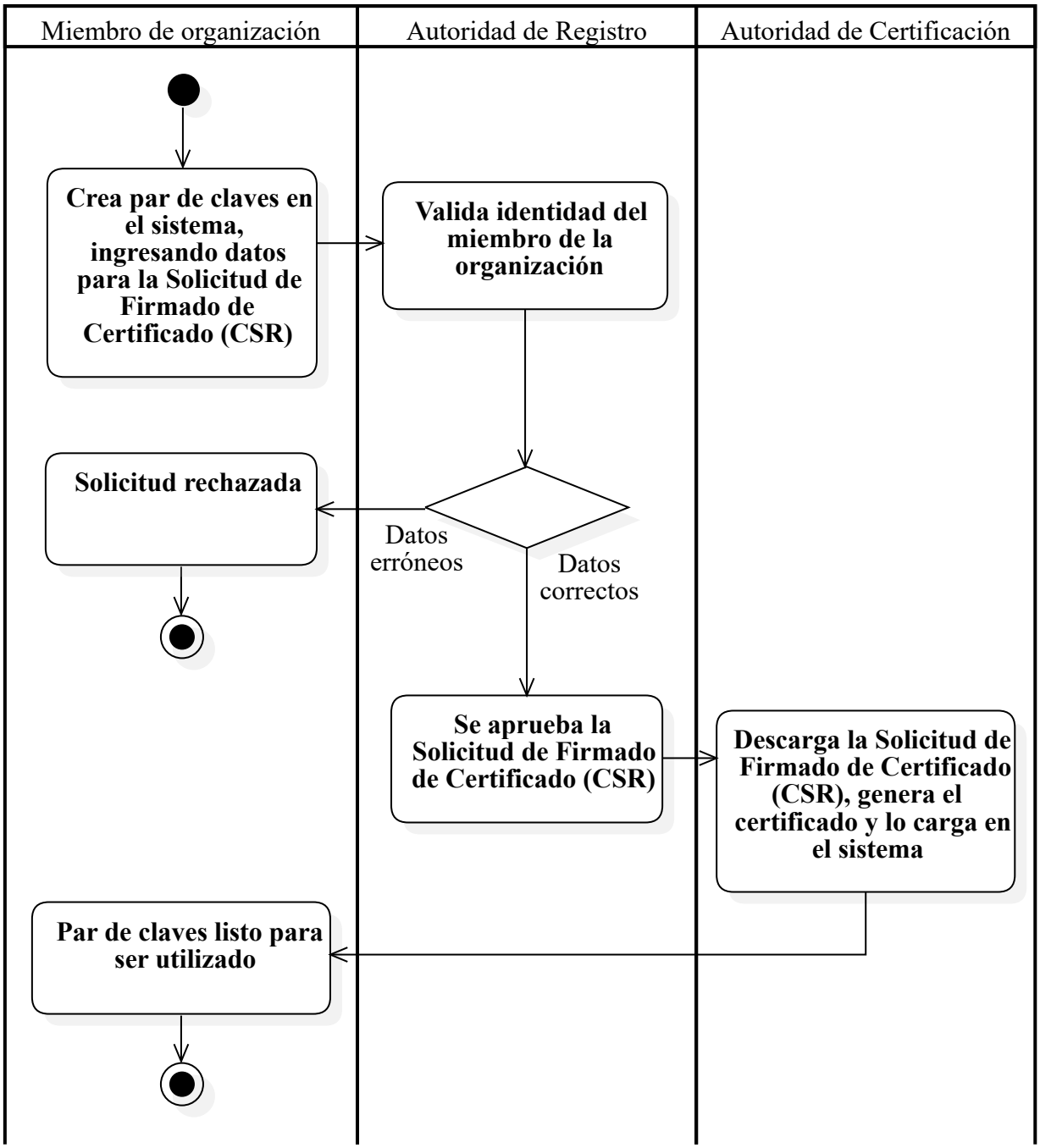

Figura 4.38: Diagrama de Actividad para la creación de par de claves y creación de Solicitud de Firmado de Certificado (CSR)

4.3.7.2. Seguridad de la base de datos MySQL

Realizar una correcta configuración del motor de base de datos MySQL que empleará el Sistema de Firma Electrónica es una tarea crucial para garantizar la seguridad en el almacenamiento de las claves de los usuarios.

El motor de base de datos MySQL posee un sistema de usuarios y privilegios de acceso. Para la utilización del motor de base de datos, los çlientes"(ya sea software configurado para emplear

MySQL para almacenar sus datos o personas que deseen realizar tareas de administración sobre las bases de datos) deben conectarse al motor mediante un nombre de usuario y una contraseña. Para efectuar diferentes operaciones, los usuarios son asignados con los correspondientes privilegios, los cuales están basados en tres diferentes niveles de operación:

- Privilegios administrativos: Privilegios de nivel global, no son específicos a una determinada base de datos del servidor. Los privilegios administrativos asignados a un determinado usuario del motor de base de datos asignan la capacidad de efectuar operaciones sobre todas las bases de datos existentes en el servidor.
- Privilegios de bases de datos: Privilegios asignados a los usuarios para la realización de operaciones sobre una determinada base de datos.
- Privilegios de objetos de bases de datos: Privilegios asignados para efectuar operaciones sobre elementos individuales de una base de datos, tales como tablas, índices, vistas, etc.

En el motor MySQL, a su vez, los usuarios pueden ser restringidos para conectarse desde ubicaciones de red específicas (hosts específicos) o para conectarse únicamente de forma local.

Usuarios del motor MySQL que empleará el Sistema de Firma Electrónica Es altamente recomendable que todos los usuarios del motor MySQL estén restringidos a conectarse únicamente de forma local (esto es, conexiones realizadas en el mismo servidor y sistema operativo que está ejecutando el motor MySQL). De esta forma, no podrán haber accesos externos a ningún objeto del motor.

A su vez, deberá existir únicamente un solo usuario "root" (usuario con privilegios administrativos totales del motor MySQL), cuyas credenciales de acceso deberán ser adecuadamente manipuladas por el personal encargado de administrar el servidor. Todos los demás usuarios del motor MySQL poseerán privilegios para efectuar operaciones únicamente sobre las bases de datos individuales para las cuales fueron creados.

Privilegios sobre la base de datos propia del Sistema de Firma Electrónica La base de datos propia del Sistema de Firma Electrónica poseerá un solo usuario con privilegios totales de datos (SELECT, INSERT, UPDATE, DELETE), cuya utilización será de forma exclusiva por el Servidor WebAPI del Sistema de Firma Electrónica. Tal como fue mencionado anteriormente, este usuario estará restringido a conectarse al motor MySQL de forma local (teniendo en cuenta que el Servidor WebAPI y el motor MySQL estarán en el mismo servidor).

4.3.7.3. Protección del servidor mediante Firewall

La implementación de un Firewall en el servidor en el que funcionará el Sistema de Firma Electrónica es indispensable para evitar accesos de intrusos desde la red. Teniendo en cuenta que el Sistema de Firma Electrónica puede ser implementado en el servidor de una organización que posea otras aplicaciones funcionando en el mismo, es tarea vital del Administrador del servidor configurar correctamente un Firewall de manera que los accesos desde la red al servidor puedan efectuarse únicamente mediante los puertos que las aplicaciones instaladas y debidamente identificadas estén escuchando. De esta forma, la única comunicación de la red con el servidor se realizará a través de puertos debidamente identificados.

Tal como se detalla en el Diagrama de Despliegue de la figura 4.1, el Sistema de Firma Electrónica se "expone" a la red únicamente mediante el puerto 8080 (puerto comúnmente utilizado para comunicaciones HTTPS, seguras). Es a través de este puerto que el Sistema de Firma

Electrónica "escucha" las solicitudes que se realizan en el Servidor WebAPI y entrega las respuestas pertinentes. Este puerto es el que deberá estar configurado en el Firewall del servidor para permitir el correcto funcionamiento del Sistema de Firma Electrónica.

4.3.7.4. Seguridad en la comunicación entre la aplicación cliente y el servidor

Tal como fue detallado en el Diagrama de Despliegue de la figura 4.1 (página 48), las comunicaciones entre la aplicación cliente y el servidor se realizan empleando el protocolo ***HTTPS***.

Sintetizando los contenidos detallados en la publicación RFC 2818: HTTP Over TLS[24], HTTPS es un protocolo basado en el clásico HTTP, que utiliza por sobre la capa de transporte TCP/IP una capa de seguridad empleando el protocolo SSL/TLS (siendo TLS el protocolo "sucesor" de SSL). Una esquematización de la diferencia entre HTTP y HTTPS puede observarse en la siguiente figura:

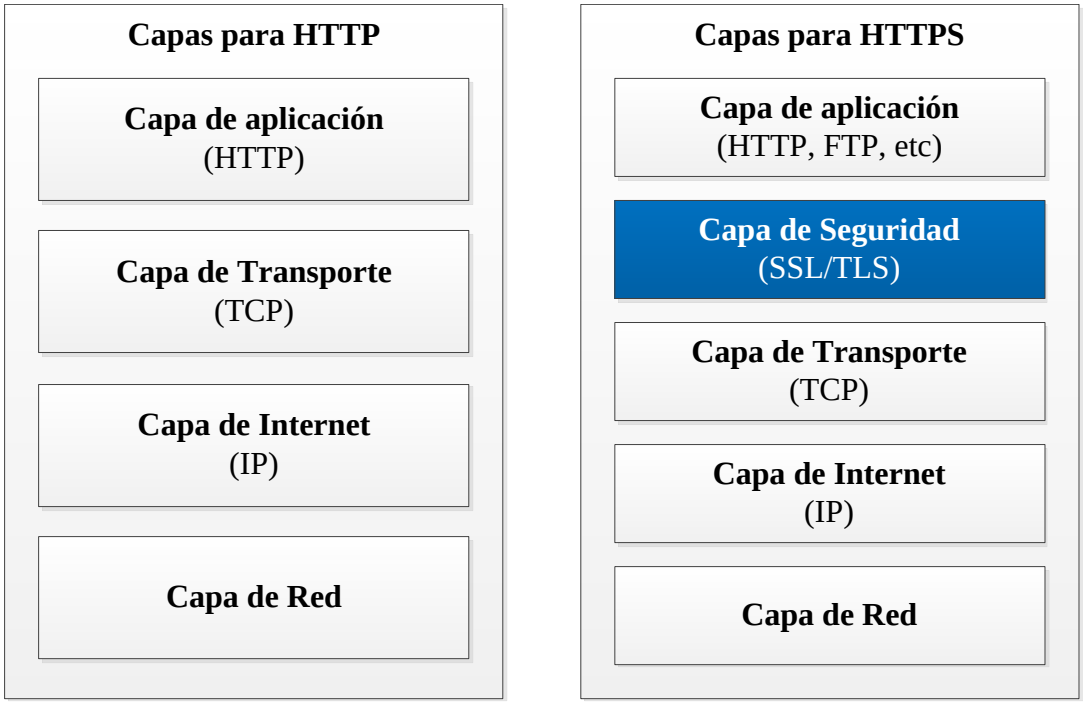

Figura 4.39: Capas empleadas en los protocolos HTTP y HTTPS

En HTTPS, empleando una capa de seguridad SSL/TLS, las comunicaciones entre el cliente (por ejemplo, un usuario empleando el explorador web Google Chrome) y el servidor se realizan de forma encriptada. Para su funcionamiento, el servidor web emplea un par de claves certificadas por una Autoridad Certificante en la que el cliente confía.

El Servidor WebAPI del Sistema de Firma Electrónica, al ser implementado para utilizar el protocolo HTTPS, debe ser proporcionado de un **par de claves y un certificado** al momento de su despliegue. La organización que implemente el Sistema de Firma Electrónica deberá adquirir un certificado apto para SSL/TLS en alguna Autoridad Certificante.

Por lo general, la adquisición de estos certificados en una Autoridad Certificante de primer nivel (tal como DigiCert [25], GeoTrust [26], GoDaddy [27]) representan un gran costo económico. Para los propósitos del presente proyecto, optaremos por emplear un certificado SSL/TLC provisto

[24] E. Rescorla. 2000. *RFC 2818: HTTP Over TLS.*
[25] https://www.digicert.com
[26] https://www.geotrust.com
[27] https://ar.godaddy.com/web-security/ssl-certificate

por la empresa ***Let's Encrypt*** [28], que ofrece certificados SSL/TLC gratuitos con duración de 90 días.

4.3.7.5. Análisis de vulnerabilidades del servidor y tests de penetración

Una correcta configuración de seguridad del servidor que hospede el Sistema de Firma Electrónica es un factor crítico a la hora de efectuar el despliegue del mismo en la organización que lo implemente. Por esta razón es que, en conjunto con el resto de las consideraciones de seguridad para el despliegue que fueron detalladas en la presente sección, se solicita a la organización implementadora que efectué un completo análisis de vulnerabilidades en su servidor, contratando a un experto o a una empresa que brinde servicios de Hacking Ético y realice análisis de seguridad.

4.3.8. Análisis de posibles vulnerabilidades de la implementación

Habiendo establecido previamente las medidas de seguridad utilizadas en la implementación, y las consideraciones recomendadas para el despliegue, en esta sección nos enfocaremos en analizar los factores de riesgo propios del diseño del Sistema de Firma Electrónica. Es decir, analizar con qué "facilidad" se encontraría un determinado atacante si logra superar las barreras de seguridad impuestas en el servidor para efectuar las siguientes operaciones de riesgo:

- Ingresar en el sistema con credenciales de otro usuario
- Generar un Token de Autenticación falso
- Acceder a la base de datos del sistema y obtener una determinada Clave Privada

Ingresar en el sistema con credenciales de otro usuario Para que un intruso ingrese en el sistema con credenciales de otro usuario, deberá disponer del nombre de usuario, la contraseña, y un código TOTP válido durante los siguientes 30 segundos (o el dispositivo móvil del usuario).

Generar un Token de Autenticación falso Para generar un Token de Autenticación falso que permita al usuario ingresar en el sistema sin utilizar credenciales de autenticación, deberá romper la clave de 128 bits que el sistema emplea para generar los Tokens mediante el esquema HMAC-SHA256, **antes de que finalice el día** (recordemos que, tal como se especificó en la sección 4.3.6.4, el sistema renueva la clave para generación de Tokens a diario). Esta tarea es totalmente imposible para las capacidades computacionales actuales.

Acceder a la base de datos del sistema y obtener una determinada Clave Privada En caso de que el usuario tenga acceso a la base de datos del sistema, solo tendrá acceso a las Claves Privadas encriptadas. Para desencriptar *cada* Clave Privada, deberá romper una encriptación AES-256. Bajo las capacidades computacionales actuales, es una tarea totalmente imposible.

[28] https://letsencrypt.org/

Capítulo 5

Resultados y verificación experimental

Habiendo desarrollado completamente el Sistema de Firma Electrónica detallado en el capítulo 4, en el presente capítulo analizaremos la solución obtenida realizando operaciones de creado de pares de claves, verificaciones de Solicitudes de Firmado de Certificado (simulando ser una Autoridad de Registro), caga de certificados firmados (simulando ser una Autoridad Certificante), y finalmente las operaciones núcleo del presente proyecto de grado: Generación de una Firma Electrónica de un archivo, y la verificación de la misma.

Para simular la utilización del Sistema de Firma Electrónica en una "Organización de Prueba" que posee su propia Infraestructura de Clave Pública (esto es, un área que hace las veces de Autoridad Certificante, con su par de claves y certificado raíz, y un área designada designada como Autoridad de Registro) debemos instalar en un equipo de prueba el software necesario para operar como Autoridad Certificante, creando un par de claves pública y privada, y un certificado "auto-firmado", propiedad de la Autoridad Certificante de esta organización ficticia. Para los propósitos de las tareas criptográficas que se realizan como Autoridad Certificante en esta verificación experimental, emplearemos el software OpenSSL[1] en un entorno Ubuntu Server 17.04 (una de las distribuciones de Linux orientada a servidores más utilizadas). Los detalles de cómo fue instalado y configurado OpenSSL son profundizados en el Apéndice A.

Utilización del Sistema de Firma Electrónica Al ingresar en el Sistema de Firma Electrónica utilizando un Usuario Normal (es decir, un usuario que emplea el sistema apara generar firmas electrónicas), podemos observar la página web mostrada en la figura 5.1, donde se listan las claves que posee el mismo en el sistema.

Para crear un nuevo par de claves en el Sistema de Firma Electrónica, debemos ingresar en la opción correspondiente del menú principal y completar los datos, tal como se observa en la figura 5.2. Los datos ingresados en el formulario conformarán la Solicitud de Firmado de Certificado (Certificate Signing Request) que deberá verificar la Autoridad de Registro, y de ser válida, certificará la Autoridad Certificante.

Una vez creado nuestro nuevo par de claves, junto con su Solicitud de Firmado de Certificado, podemos verlo listado en la lista de claves, en la sección "Claves pendientes de certificación" (Figura 5.3). La clave privada generada se encuentra almacenada de forma segura en la base de datos del Sistema de Firma Electrónica. La clave pública derivada es la siguiente:

04 03 17 D7 DE FB 28 21 06 DD DD 1F 46 68 FF 87 48
63 2A AF 6A 33 11 D4 C9 13 6A EC 57 5E 08 36 E4
D9 52 EA D6 5E 8B A2 C3 4B 20 D1 4A 9F 24 34 19
44 92 9F 6F D1 85 A6 AA A0 48 30 67 9B A2 75 27

[1] Iain Alderman y Zach Miller. 2009. *How to Set Up a CA*. URL: http://pages.cs.wisc.edu/~zmiller/ca-howto (visitado 27-09-2017).

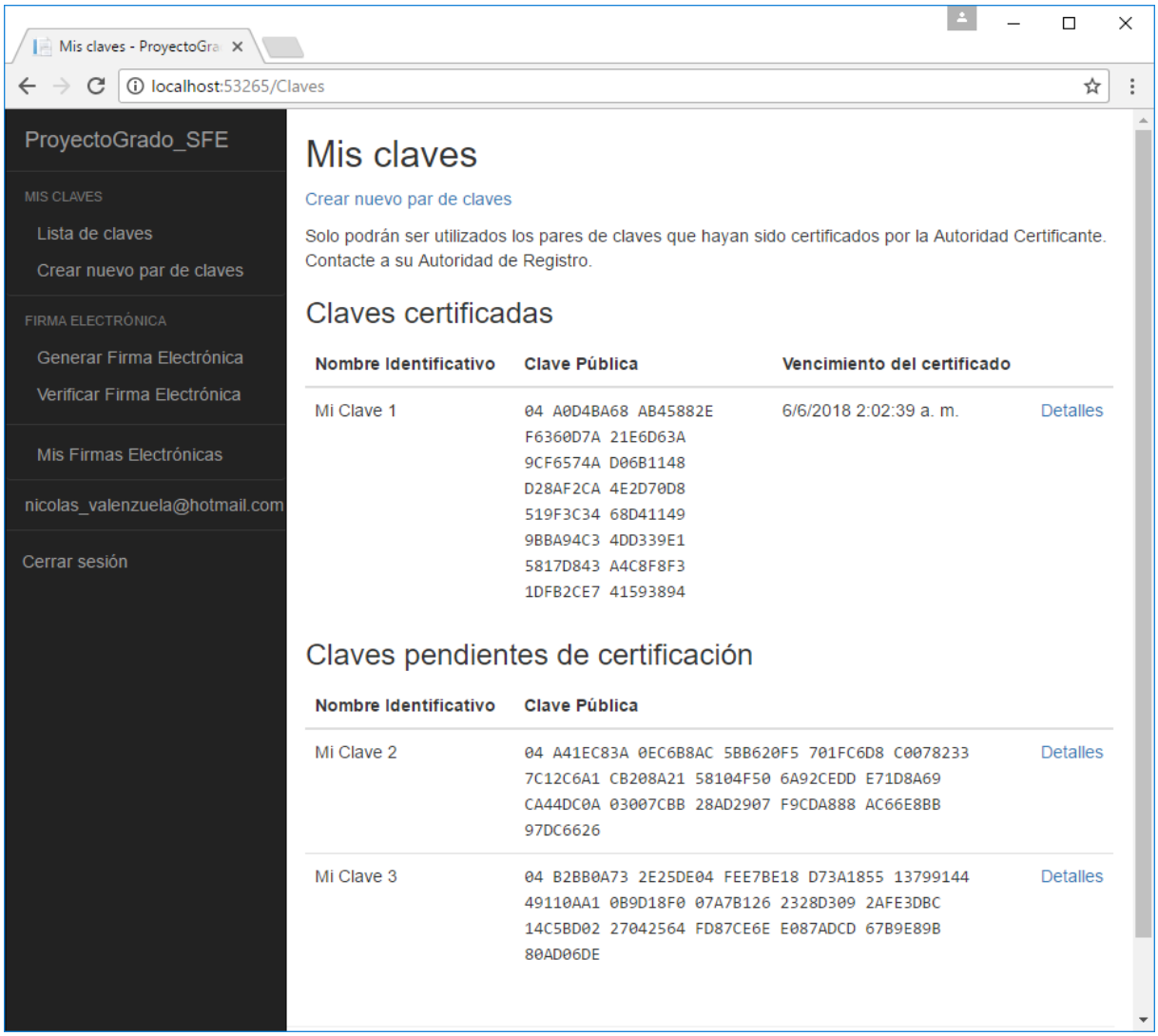

Figura 5.1: Lista de claves del usuario

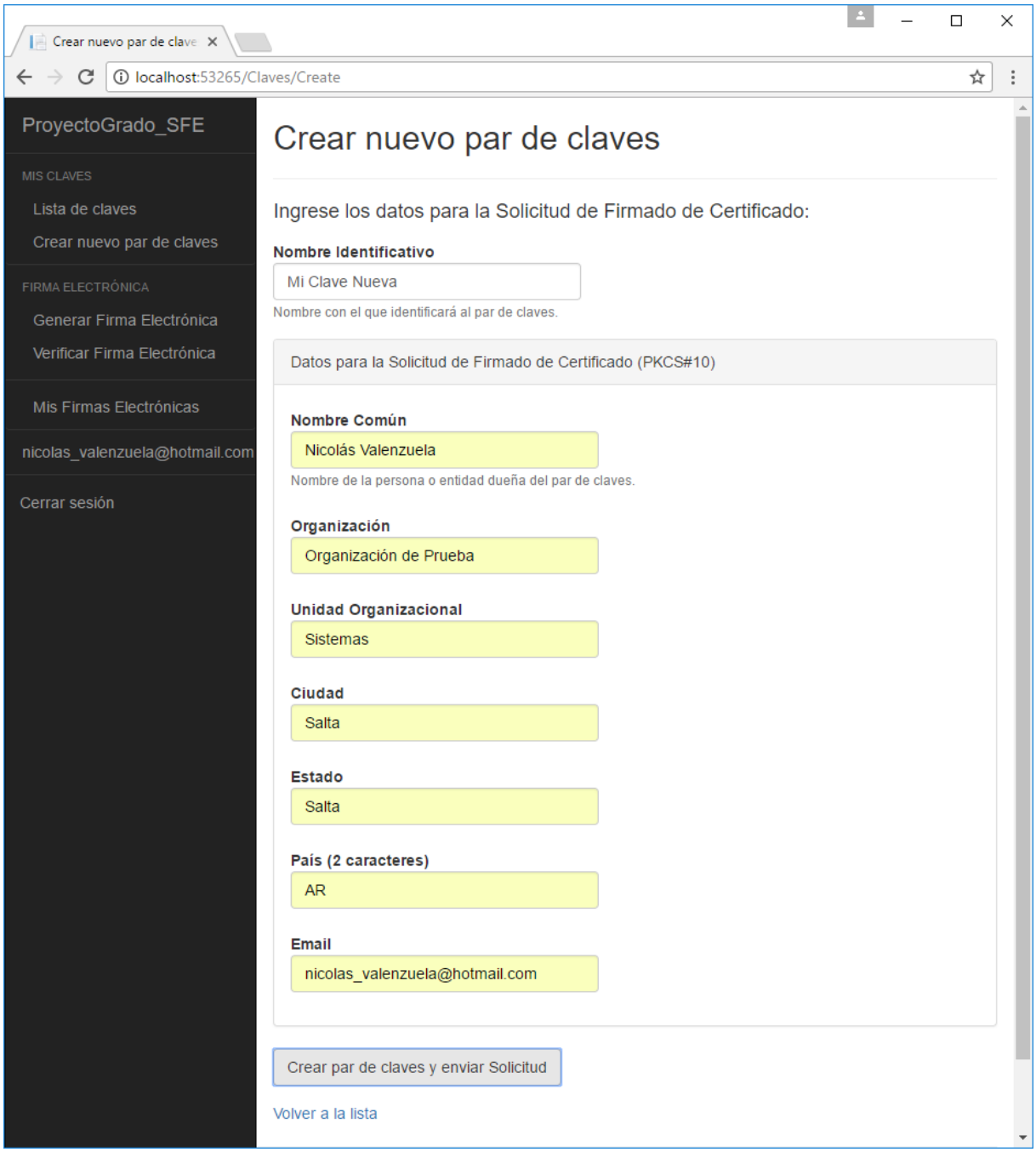

Figura 5.2: Creación de un nuevo par de claves y Solicitud de Firma de Certificado

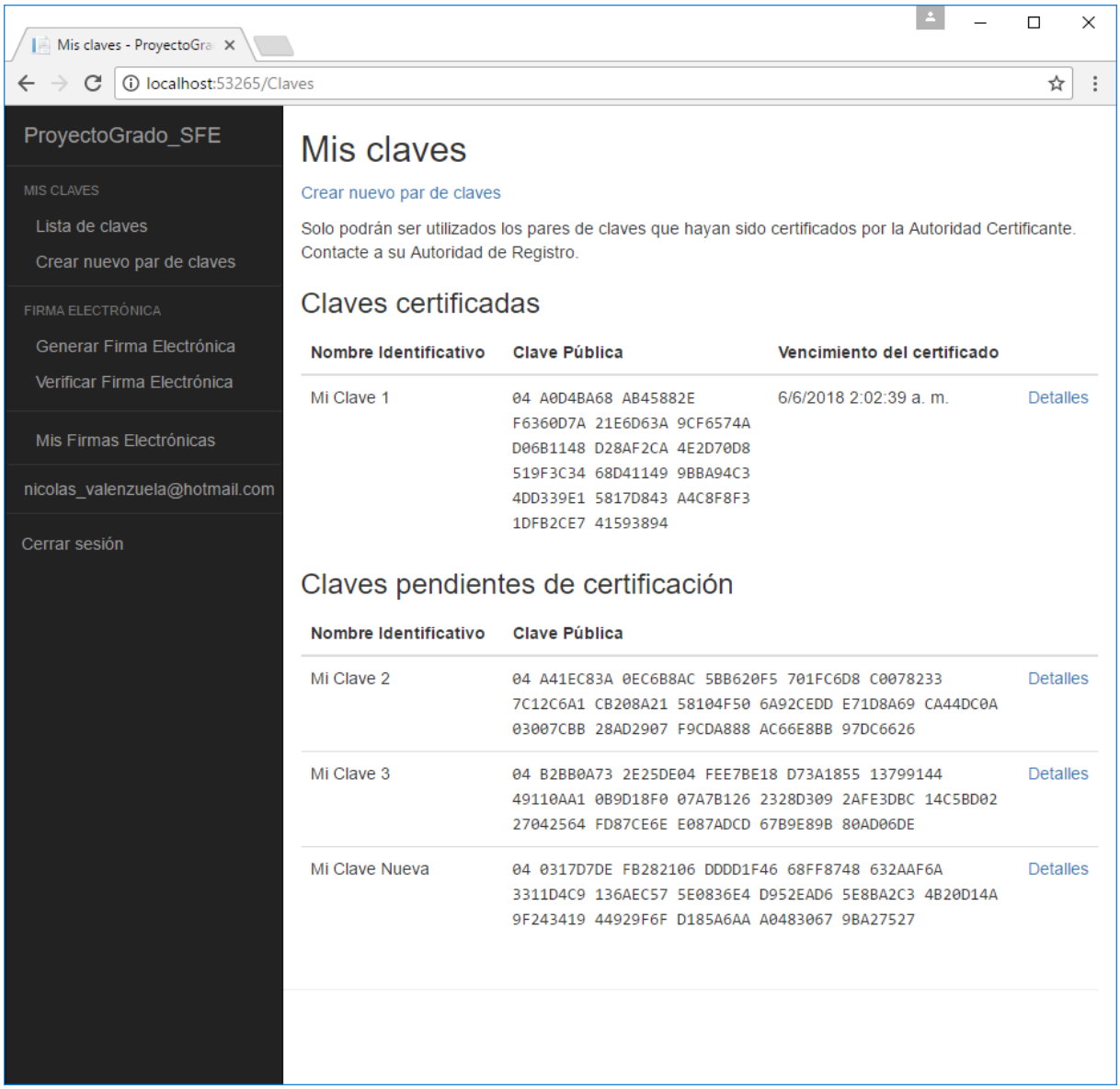

Figura 5.3: Nuevo par de claves listado en las claves del usuario

Una vez creado el nuevo par de claves, reingresamos al Sistema de Firma Electrónica, esta vez con un usuario con rol de "Autoridad de Registro". En la página principal de este usuario podemos observar la lista de claves que fueron generadas en el sistema por los usuarios, y que requieren que una Autoridad de Registro las verifique previo a la generación de su certificado (figura 5.4). Cuando el usuario de la Autoridad de Registro ingresa a ver los detalles de la clave que generamos anteriormente, puede ver la página de la figura 5.5, en donde están detallados los campos de la solicitud PKCS#10 que completó el usuario solicitante. Una vez verificados los datos como correctos, hace click en "Aprobar".

A continuación, reingresamos al Sistema de Firma Electrónica empleando esta vez el usuario de la Autoridad Certificante, mediante el cual descargaremos la Solicitud de Firmado de Certificado y generaremos el certificado correspondiente. Al ingresar al sistema como Autoridad Certificante, podemos observar una lista de las claves aprobadas por la Autoridad de Registro, que están a la espera de ser certificadas (figura 5.7). Elegimos la clave creada anteriormente (la que tiene Nombre Identificativo "Mi Clave Nueva") para ver sus detalles. En la página de la figura 5.8 la Autoridad Certificante puede observar qué usuario generó la solicitud, los datos de la aprobación de la Autoridad de Registro, y los campos completados por el usuario de la Solicitud de Firmado de Certificado PKCS#10. En esta misma página, la Autoridad Certificante descarga el archivo de la Solicitud de Firmado de Certificado (el archivo descargado se llama ***request_Nicolás_Valenzuela_Sistemas.csr***), el cual empleará para firmar y generar el certificado. Para hacer esto, habiendo configurado el software OpenSSL para operar como Autoridad Certificante tal como se detalla en el Apéndice A, empleamos el siguiente comando de OpenSSL:

```
$> openssl ca –out certificado_Nicolas_Valenzuela_Sistemas.crt
   –infiles request_Nicolás_Valenzuela_Sistemas.csr
```

Ejecutando este comando, obtenemos la siguiente salida, que nos indica que el certificado ***certificado_Nicolas_Valenzuela_Sistemas.crt*** fue generado correctamente:

```
$>

Using configuration from /usr/local/ssl/openssl/openssl.cnf
Loading 'screen' into random state – done
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
organizationName      :ASN.1 12:'Organización de Prueba'
organizationalUnitName:ASN.1 12:'Sistemas'
localityName          :ASN.1 12:'Salta'
stateOrProvinceName   :ASN.1 12:'Salta'
countryName           :PRINTABLE:'AR'
commonName            :ASN.1 12:'Nicolás Valenzuela'
Certificate is to be certified until Jun  7 04:43:03 2018 GMT
   (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

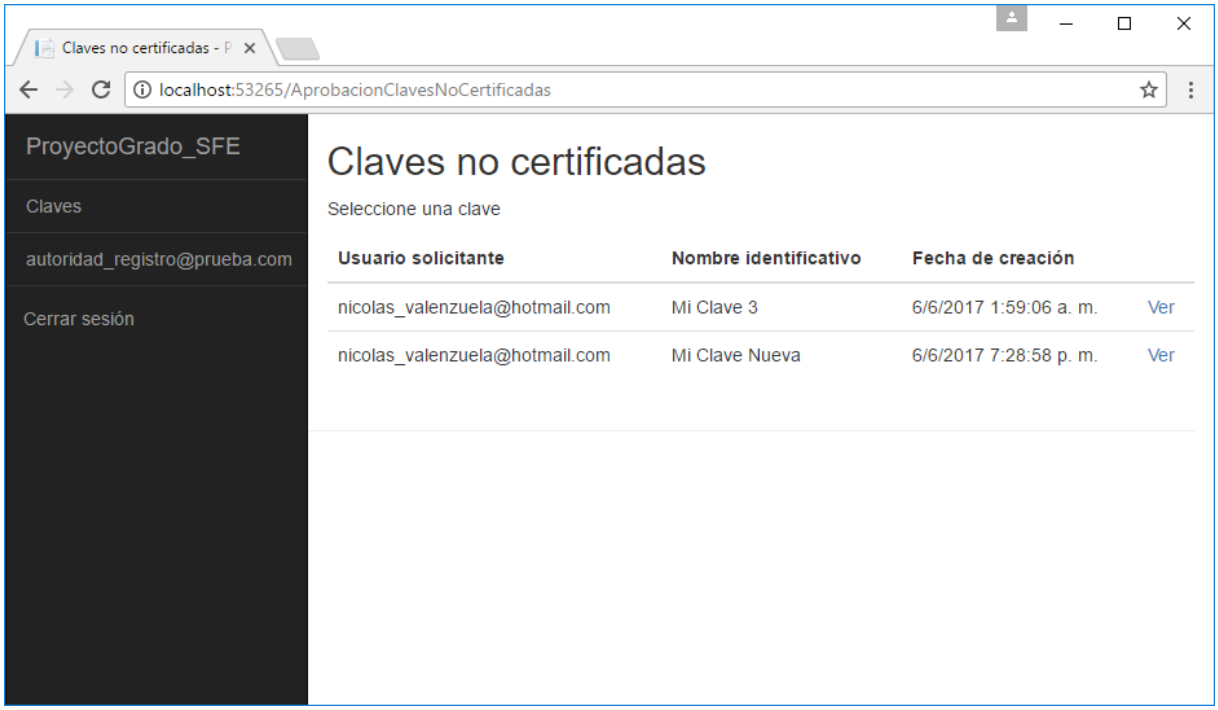

Figura 5.4: Autoridad de Registro: Lista de claves pendientes de verificación

Figura 5.5: Autoridad de Registro: Revisión de detalles de Solicitud de Firmado de Certificado

En la figura 5.6 se puede observar una captura de pantalla del certificado que acabamos de generar.

Una vez que fue generado correctamente el certificado mediante el comando previamente mencionado, procedemos a cargarlo en la misma página donde descargamos la Solicitud de Firmado de Certificado (figura 5.8). De esta forma, la clave está lista para ser utilizada por su dueño para generar Firmas Electrónicas. Volviendo a ingresar con nuestro Usuario Normal de prueba, podemos observar que nuestra clave se encuentra listada en la sección "Claves Certificadas" (figura 5.9). Si ingresamos a ver los detalles de la clave que acaba de ser certificada, encontraremos listados los detalles de la verificación por parte de la Autoridad de Registro y la certificación de la Autoridad Certificante (figura 5.10). A su vez, podremos descargar el certificado que generó la Autoridad Certificante.

A continuación, podemos ingresar en la opción del menú principal "Generar Firma Electrónica", para generar la firma de un archivo de prueba. Tal como se puede observar en la figura 5.11, debemos elegir la clave que creamos (pudiendo encontrarla por su Nombre Identificativo "Mi Clave Nueva"), y seleccionamos el archivo a firmar. En este caso, el archivo a firmar es ***test.txt***, cuyo contenido es el siguiente:

```
Este es un texto de prueba, para verificar el funcionamiento del Sistema de
    Firma Electrónica.
```

Al hacer click en el botón "Generar Firma Electrónica", descargaremos el archivo ***test.txt.frm***, que contiene la firma electrónica propiamente dicha y el certificado de la clave empleada para generarla. Teniendo posesión del archivo ***test.txt*** y ***test.txt.frm***, debemos transmitirlos juntos a nuestro destinatario para que pueda verificar su integridad, autenticidad y autoría. Para simular la validación del archivo contra su Firma Electrónica, ingresamos en la opción "Verificar Firma Electrónica", donde veremos la página de verificación en la que cargaremos el archivo a verificar (***test.txt***) y su firma electrónica ***test.txt.frm*** (figura 5.12). Cargamos los archivos en los campos correspondientes y hacemos click en "Verificar", para ver el resultado de la validación.

En la página de resultado, que podemos observar en la figura 5.13, se muestra un mensaje indicando que el archivo superó la validación con su firma electrónica a la vez que podemos observar la información del certificado incluido, para reconocer quién es el firmante.

Si modificamos el texto contenido en el archivo ***test.txt*** y volvemos a realizar la verificación de la firma electrónica, esta fallará y podremos observar un mensaje indicando ello, tal como se detalla en la figura 5.14.

Finalmente, ingresando en la opción del menú "Mis Firmas Electrónicas" encontraremos listadas las firmas electrónicas generadas por el usuario actual, indicando nombres de los archivos, valor Hash, fecha de generación y clave que fue utilizada (figura 5.15).

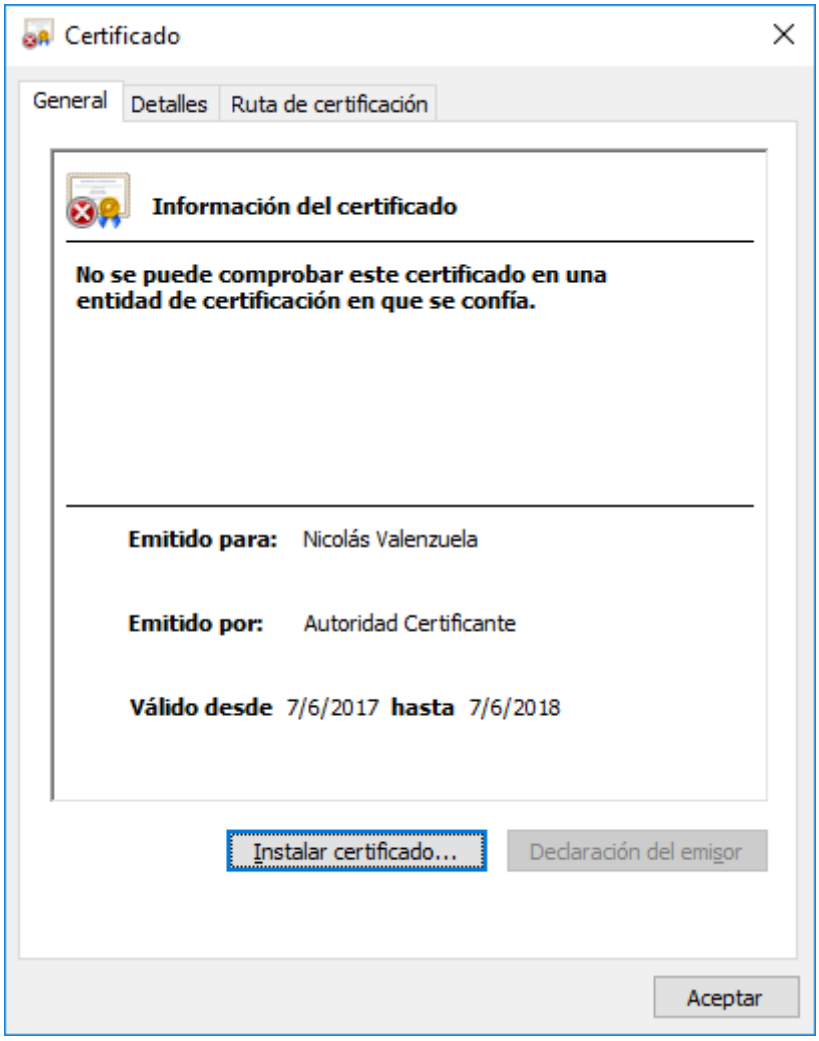

Figura 5.6: Certificado generado por la Autoridad Certificante

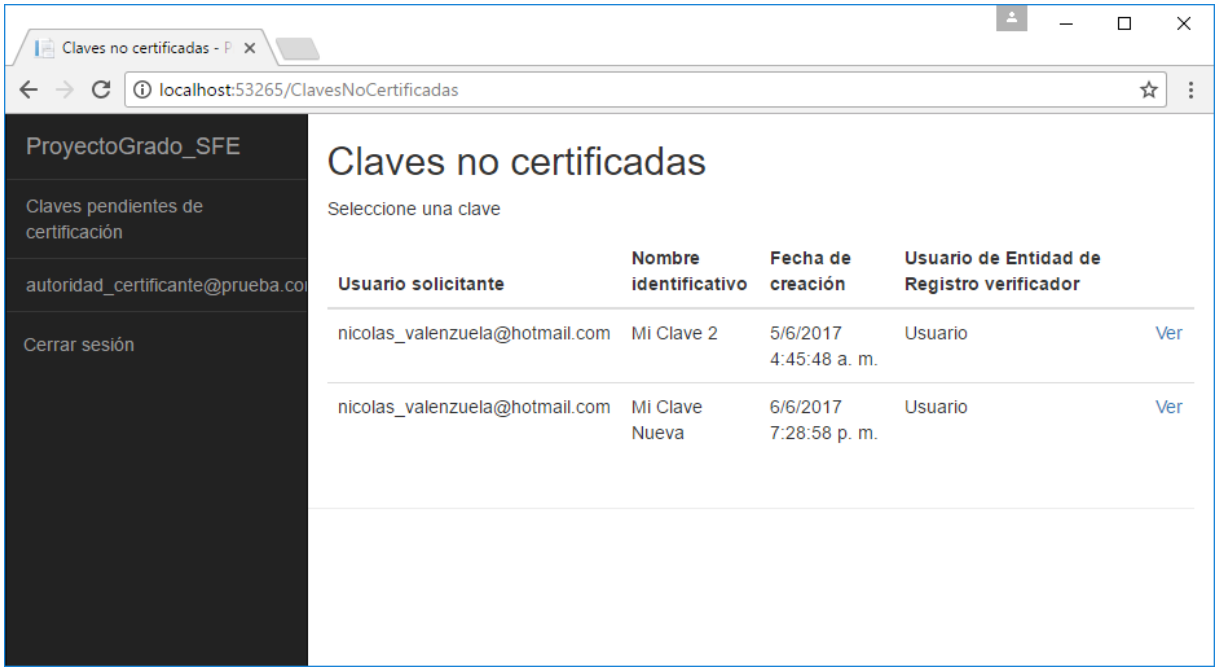

Figura 5.7: Autoridad Certificante: Lista de claves aprobadas pendientes de certificación

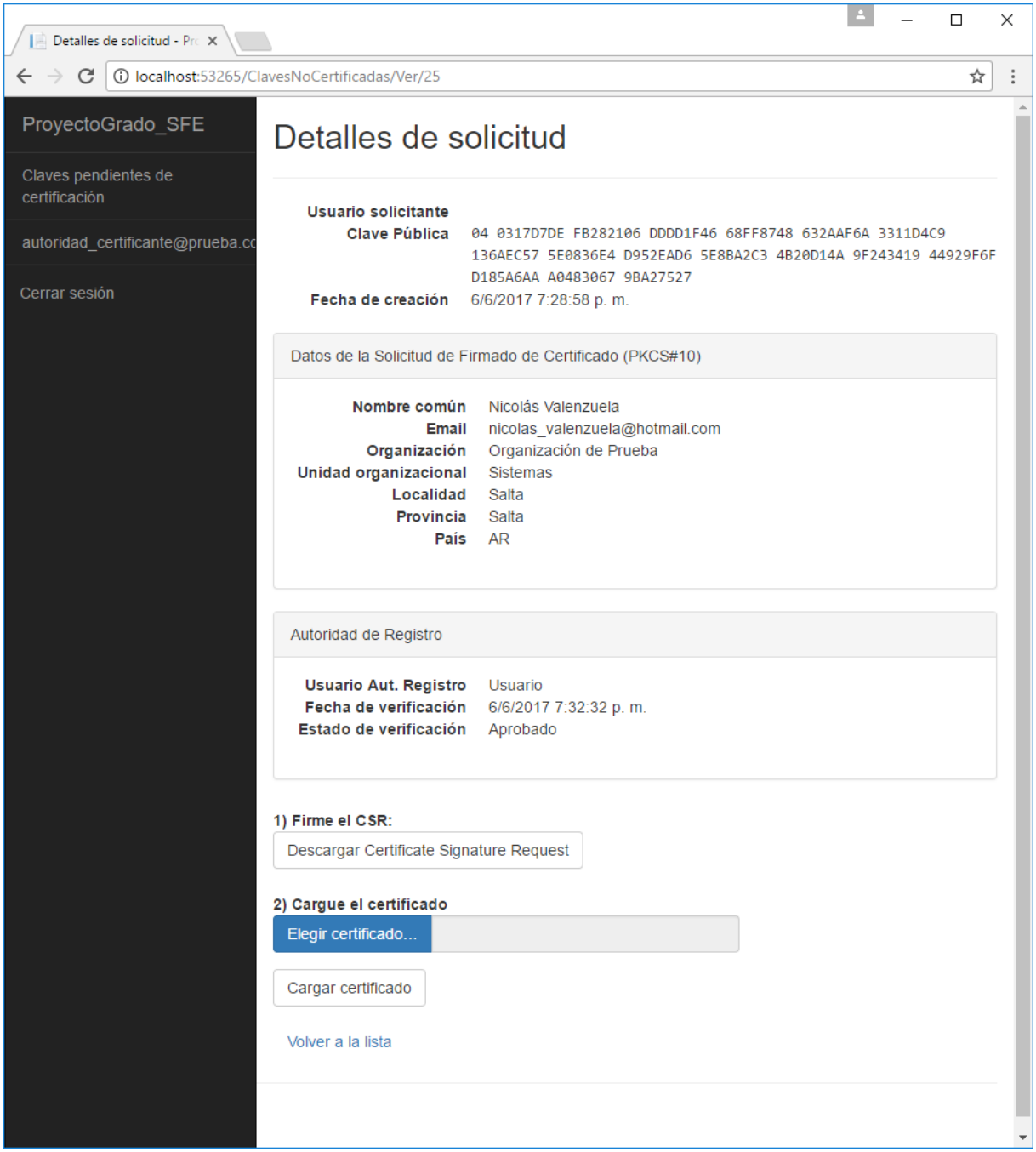

Figura 5.8: Autoridad Certificante: Revisión de detalles de Solicitud de Firmado de Certificado y descarga de CSR

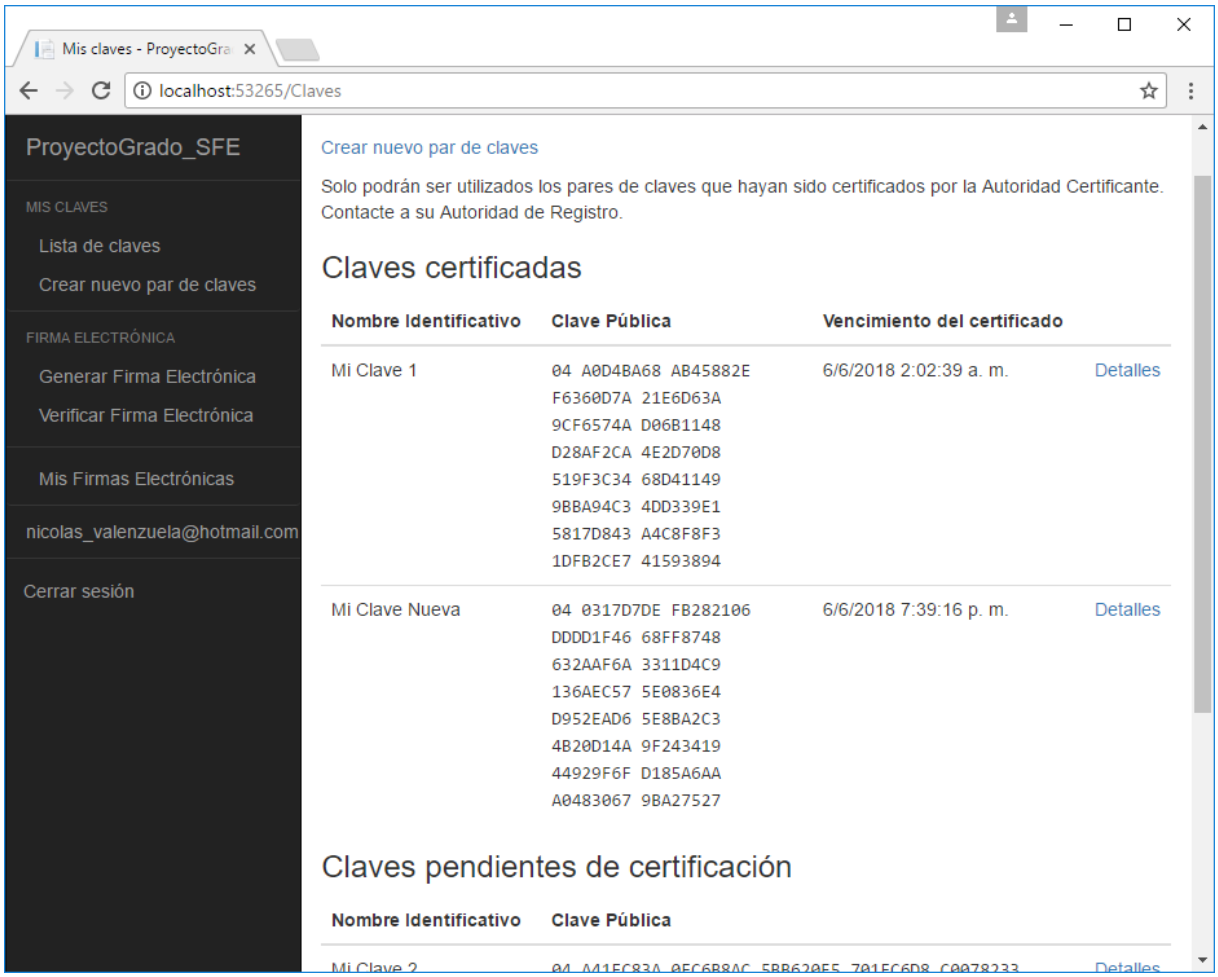

Figura 5.9: Clave certificada y lista para ser utilizada por el usuario

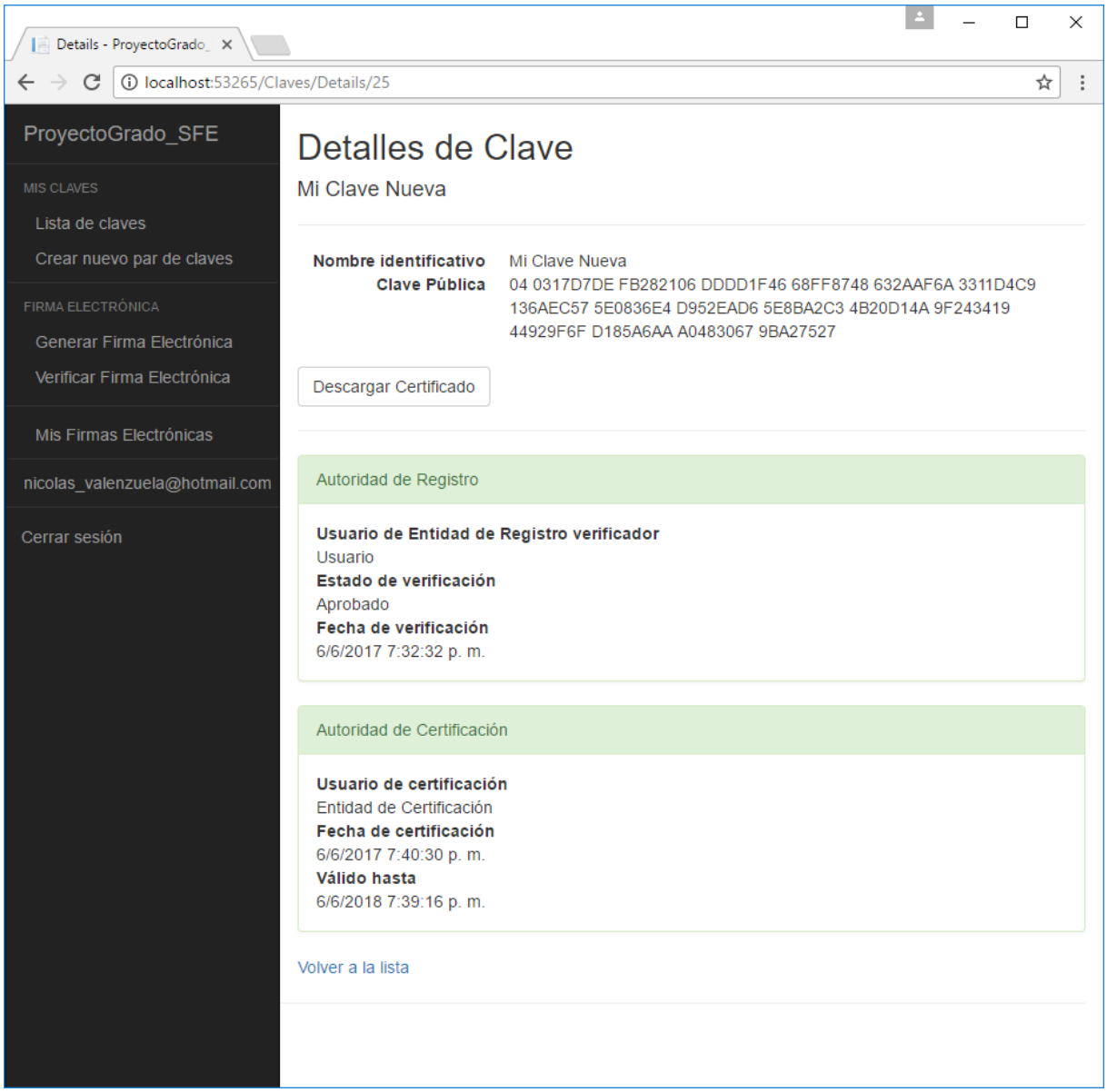

Figura 5.10: Detalles de la clave certificada

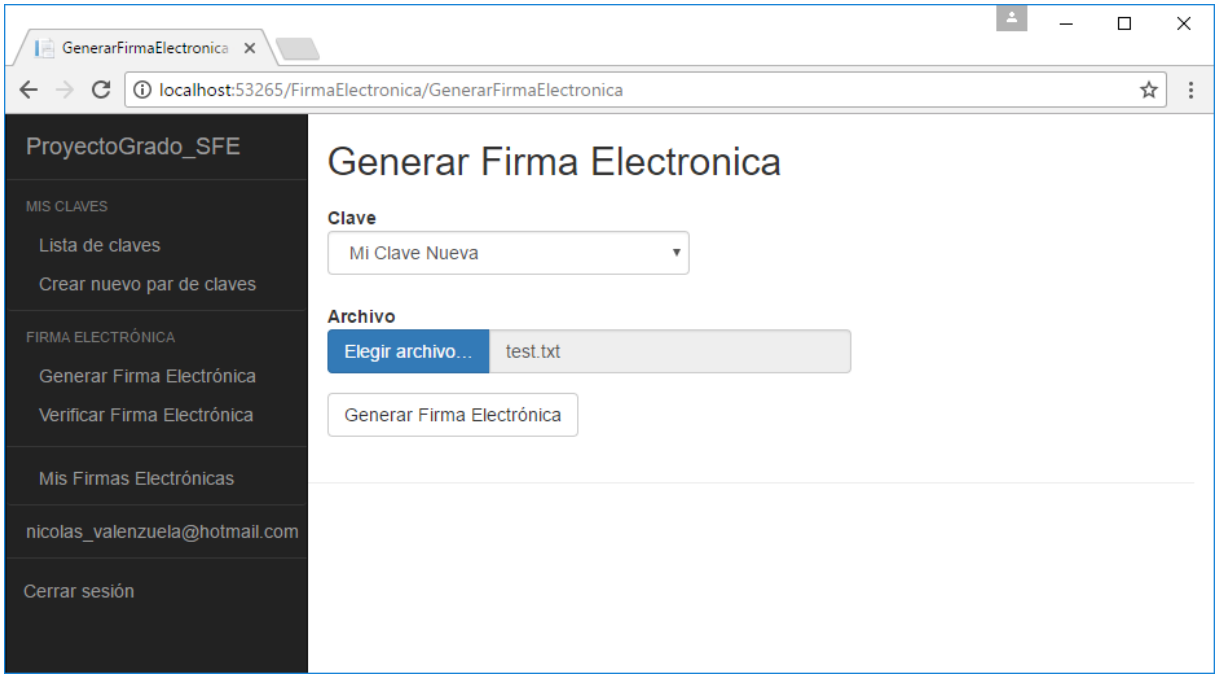

Figura 5.11: Generación de Firma Electrónica

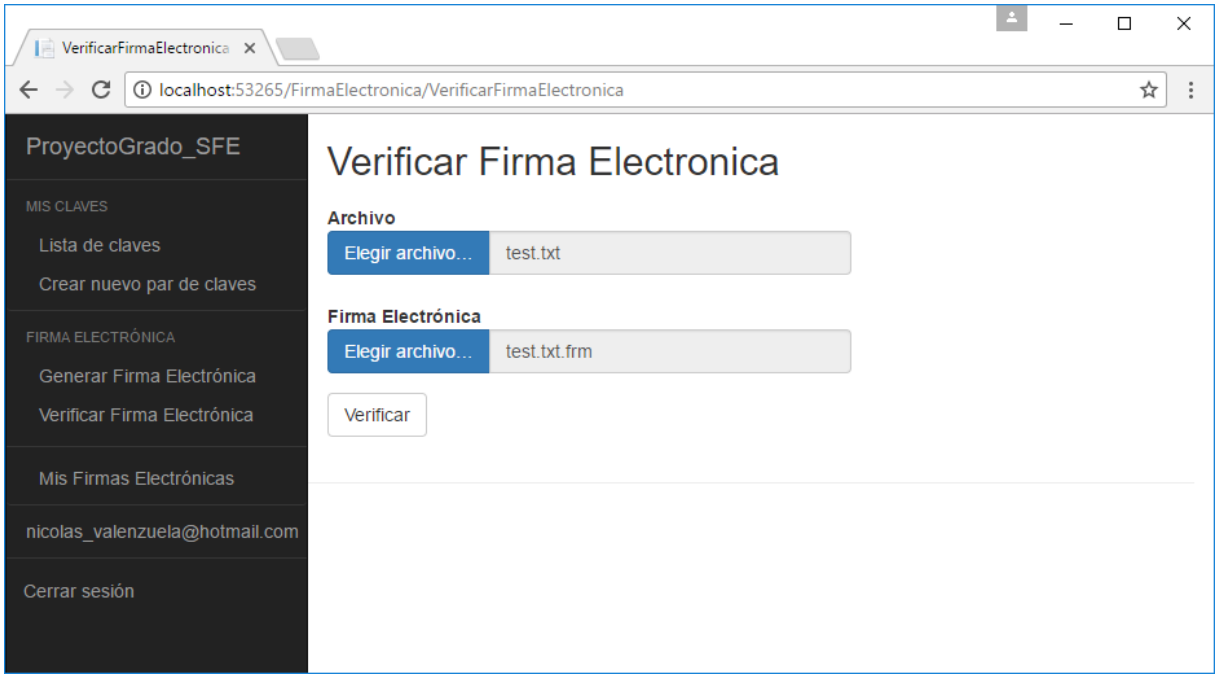

Figura 5.12: Verificación de Firma Electrónica

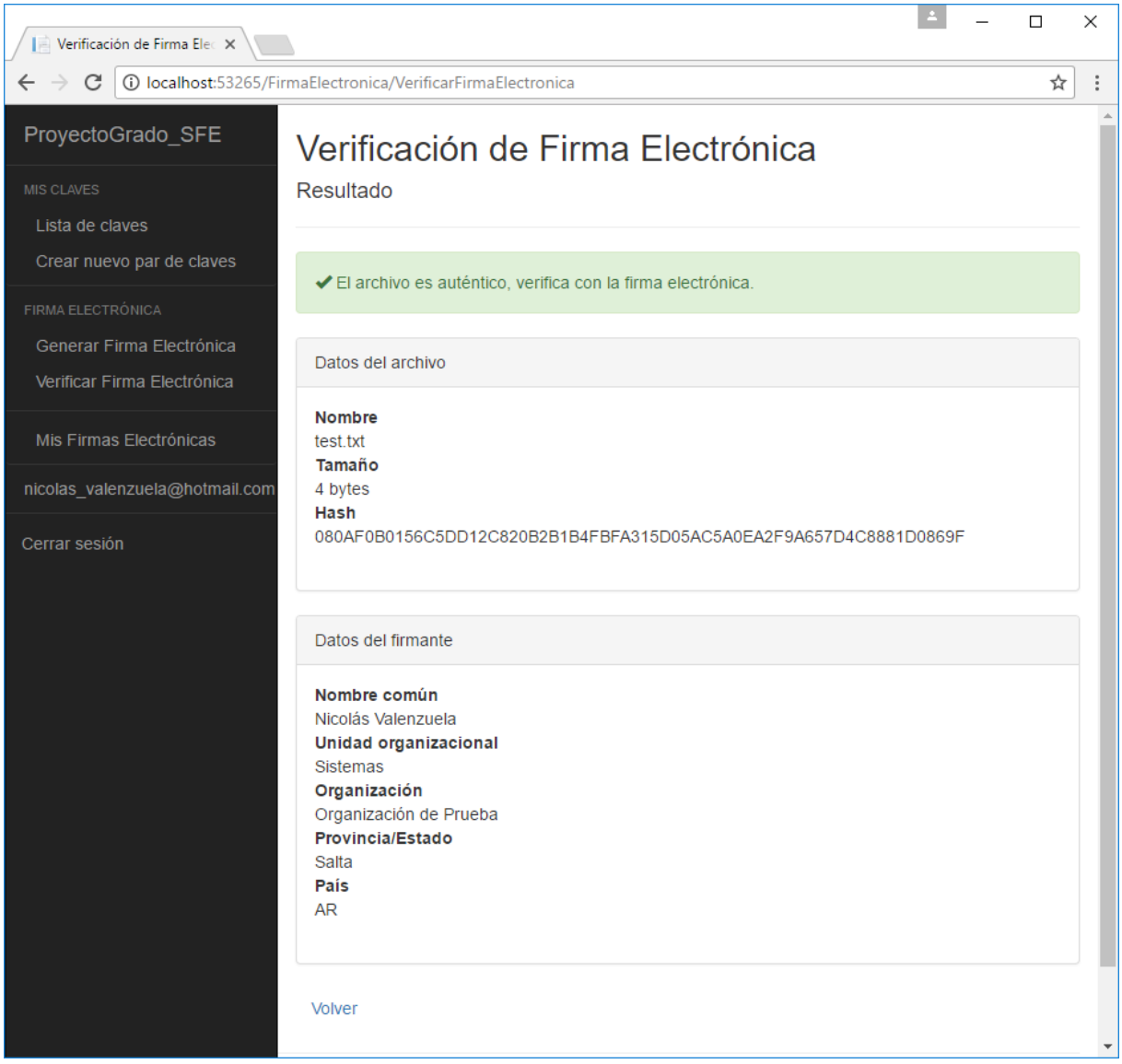

Figura 5.13: Resultado de la Verificación de Firma Electrónica: Correcto

Figura 5.14: Resultado de la Verificación de Firma Electrónica: Erróneo

Figura 5.15: Registro de Firmas Electrónicas

Capítulo 6

Conclusión

La Criptografía aplicada a las Ciencias Informáticas es un área de trabajo muy extensa. La aplicación de conceptos criptográficos a cualquier proyecto informático, con el objetivo de brindar ciertas capas de seguridad al uso del mismo, es una empresa que requiere un análisis a conciencia de numerosas especificaciones, esquemas y algoritmos.

Diseñar un Sistema de Firma Electrónica con Encriptación de Curva Elíptica, y en particular su Librería Criptográfica, es un proyecto de implementación compleja, requiriendo una laboriosa tarea de diseño previo que demandó la lectura y análisis de muchísima documentación sobre esquemas criptográficos, evaluación de alternativas y realización de pruebas, con el objetivo de producir un sistema seguro y óptimo, que pueda ser realmente empleado en la práctica.

En la actualidad hay un amplio catálogo de librerías y frameworks en diversos lenguajes de programación que simplifican la tarea de implementar criptografía en proyectos informáticos. El enfoque del presente proyecto fue realizar un desarrollo prescindiendo de librerías existentes, para cumplir un objetivo "secundario": Implementar uno mismo los fundamentos matemáticos inherentes a la criptografía de curvas elípticas para comprender en profundidad el funcionamiento de un sistema que emplea criptografía.

Como futura línea de desarrollo, es posible complementar el proyecto desarrollando nuevas experiencias para los usuarios empleando otras plataformas además de la web: Plataformas móbiles y aplicaciones de escritorio. El presente proyecto de grado tuvo un enfoque más intensivo en el diseño e implementación del sistema del lado del Servidor.

Bibliografía

Agence nationale de la sécurité des systèmes d'information. 2014. *Référentiel Général de Sécurité version 2.0*.

Alderman, Iain y Zach Miller. 2009. *How to Set Up a CA*. URL: http://pages.cs.wisc.edu/~zmiller/ca-howto (visitado 27-09-2017).

Asociación de Investigación en Software Inteligente. 2007. *Diseño y Construccion de Aplicaciones Agiles con el Metodo Scrum*.

Bauer, Bela et al. 2016. "Hybrid quantum-classical approach to correlated materials". En: *Physical Review X* 6.3.

Bernstein, Daniel J. y Tanja Lange. 2007. *Faster addition and doubling on elliptic curves*. Cryptology ePrint Archive, Report 2007/286.

Blake-Wilson, Simon et al. 2006. *Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS)*.

Bundesamt für Sicherheit in der Informationstechnik. 2017. *Kryptographische Verfahren: Empfehlungen und Schlusselllängen*.

Canós, José H., Patricio Letelier y María Carmen Penandés. 2003. *Metodologías Ágiles para el desarrollo de software*.

Certicom Research. 2010. *SEC 2: Recommended Elliptic Curve Domain Parameters V2*.

Hankerson, Darrel, Alfred J. Menezes y Scott Vanstone. 2010. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Inc. ISBN: 0-387-95273-X.

Honorable Congreso de la Nación Argentina. 2001. *Ley N° 25.506 de Firma Digital*.

Jones, M. et al. 2015. *RFC 7519: JSON Web Token (JWT)*.

Kaliski, B. 2000. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898 (Informational). Internet Engineering Task Force.

Larman, Craig. 2003. *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado (2da Edición)*. Pearson - Prentice Hall. ISBN: 84-205-3438-2.

LMU – Ludwig-Maximilians-Universität München, Research Unit of Programming and Software Engineering. 2016. *UWE – UML-based Web Engineering*. URL: http://uwe.pst.ifi.lmu.de/aboutUwe.html (visitado 27-09-2017).

M'Raihi, David et al. 2011. *RFC 6238-TOTP: Time-based one-time password algorithm*.

NIST (National Institute of Standards and Technology), Elaine Barker. 2016. *NIST Special Publication 800-57 Part 1 - Revision 4: Recommendation for Key Management*.

Nystrom, Magnus y Burt Kaliski. 2000. *PKCS #10: Certification Request Syntax Specification Version 1.7*. RFC 2986. DOI: 10.17487/RFC2986.

Perrin, Trevor et al. 2002. "Delegated cryptography, online trusted third parties, and PKI". En: *1nd Annual PKI Research Workshop*. Citeseer.

Rescorla, E. 2000. *RFC 2818: HTTP Over TLS*.

Rising, Linda y Norman S. Janoff. 2000. "The Scrum software development process for small teams". En: *IEEE software* 17.4, págs. 26-32.

Rotger, Llorenç Huguet, Josep Rifà Coma y Juan Gabriel Tena Ayuso. *Criptografía con curvas elípticas*.

Schwaber, Ken. 1995. "SCRUM Development Process". En: *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, págs. 117-134.

Schwaber, Ken y Jeff Sutherland. 2001. *The Scrum Guide*.

Shor, Peter W. 1999. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". En: *SIAM review* 41.2.

The Open Web Application Security Project. 2017. *OWASP Top 10 - 2017 Release Candidate - The Ten Most Critical Web Application Security Risks*.

Wuille, Pieter. 2017. *Bitcoin Network Graphs*. URL: http://bitcoin.sipa.be/ (visitado 27-09-2017).

Apéndice A

Configuración de OpenSSL en una Autoridad Certificante

En el entorno de prueba, fue instalado OpenSSL empleando el siguiente comando:

```
$> sudo apt-get install openssl
```

Para la operación de OpenSSL como Autoridad Certificante, se crearon carpetas mediante el siguiente comando:

```
$> mkdir -m 0700 \
        /root/CA \
        /root/CA/certs \
        /root/CA/crl \
        /root/CA/newcerts \
        /root/CA/private
```

Luego, se creó el archivo ***index.txt*** donde son registrados los certificados que históricamente van siendo generados, y el archivo ***serial***, que define el último número de serie a emplear para la generación de certificados:

```
$> touch /root/CA/certindex.txt
$> echo 1000 >> /root/CA/serial
```

El archivo de configuración ***openssl.cnf*** ubicado en ***/usr/local/ssl/openssl.cnf*** que empleamos, es el siguiente:

```
#
# OpenSSL configuration file.
#

# Establish working directory.

dir                     = /root/CA

[ ca ]
default_ca              = CA_default

[ CA_default ]
serial                  = $dir/serial
database                = $dir/certindex.txt
```

```
new_certs_dir   = $dir/certs
certificate             = $dir/certs/ca.crt
private_key             = $dir/private/raiz_claveprivada.pem
default_days    = 365
default_md              = md5
preserve                = no
email_in_dn             = no
nameopt                 = default_ca
certopt                 = default_ca
policy                  = policy_match

[ policy_match ]
countryName                     = supplied
stateOrProvinceName             = supplied
organizationName                = supplied
organizationalUnitName          = supplied
commonName                      = supplied
emailAddress                    = supplied

[ req ]
default_bits    = 1024  # Size of keys
default_keyfile = key.pem       # name of generated keys
default_md              = md5   # message digest algorithm
string_mask             = nombstr       # permitted characters
distinguished_name      = req_distinguished_name
req_extensions          = v3_req

[ req_distinguished_name ]
# Variable name                 Prompt string
#----------------------  ----------------------------------
0.organizationName      = Organization Name (company)
organizationalUnitName  = Organizational Unit Name (department,
    division)
emailAddress            = Email Address
emailAddress_max        = 40
localityName            = Salta
#stateOrProvinceName    = Salta
countryName             = AR
countryName_min         = 2
countryName_max         = 2
commonName              = Common Name (hostname, IP, or your
   name)
commonName_max          = 64

# Default values for the above, for consistency and less typing
   .
# Variable name                         Value
#------------------------
   ------------------------------------
0.organizationName_default      = My Company
```

```
localityName_default            = My Town
stateOrProvinceName_default     = State or Providence
countryName_default             = US

[ v3_ca ]
basicConstraints        = CA:TRUE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always

[ v3_req ]
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash
```

En primer lugar, crearemos el par de claves de la curva elíptica **secP256k1**de la Autoridad Certificante de la Organización de Prueba, empleando el siguiente comando de OpenSSL:

```
$> openssl ecparam -name secp256k1 -genkey -out
   raiz_claveprivada.pem
```

Esta clave privada es copiada en /root/CA/private/raiz_claveprivada.pem. La clave privada de 256 bits generada por OpenSSL en el archivo *raiz_claveprivada.pem* es la siguiente:

FB BD 07 49 33 D8 D9 4A BF 2C 42 08 96 D9 14 7D
9C 52 49 28 43 F8 18 62 BF F4 7E BE DB E1 0C A8

Por otra parte, la clave pública derivada de esa clave privada, es la siguiente:

04 62 60 F3 28 BC 40 9B 73 F5 D7 13 D8 BC 7F 2D 77
F4 8F 64 DC 13 94 0E 2B B1 C1 71 5D C2 4F 0B 2C
FB 05 97 3C 9E B5 08 79 4B CF F8 B3 A6 FE 02 DB
38 D0 06 A9 73 6F 4C 96 5D A8 2C 0B C5 C3 2B 0C

A continuación, crearemos un certificado “auto-firmado” para conformar el par de claves y el certificado de clave pública de la Autoridad Certificante de la Organización de Prueba, empleando el siguiente comando:

```
$> openssl req -new -x509 -key raiz_claveprivada.pem -out ac.
   crt -days 730
```

De esta forma, se generó el certificado ***ac.crt*** con validez por 730 días, utilizando el par de claves previamente generado. Este certificado ***ac.crt*** es copiado en /root/CA/certs/ca.crt.

Habiendo realizado todo esto, el software OpenSSL está configurado y listo para operar como Autoridad Certificante.

Apéndice B

Implementación de la Librería Criptográfica

En el presente apéndice de pueden observar las implementaciones de la Librería Criptográfica en lenguaje C#. La compilación del ensamblado se realizó utilizando el Tipo de Salida "Biblioteca de Clases" y la plataforma destino ".NETStandard 1.4". De esta forma, se genera una librería que puede ser utilizada en plataformas Windows, u OS X.

B.1. Sub-librería de Operaciones Criptográficas

B.1.1. Clase "KeyPair"

```
using System.Numerics;

namespace LibreriaCriptografica
{
    public class KeyPair
    {
        public static KeyPair GenerarNuevo() {

            var randBigInteger = Helper.RandomPrivKey();
            PrivateKey priv = new PrivateKey(randBigInteger);
            PublicKey pub = new PublicKey(randBigInteger * Params.G);

            return new KeyPair() { _privateKey = priv, _publicKey = pub
                };
        }
        private PrivateKey _privateKey;
        private PublicKey _publicKey;

        public PrivateKey PrivateKey => _privateKey;
        public PublicKey PublicKey => _publicKey;

    }

    public class PrivateKey
    {

        public PrivateKey(BigInteger n)
        {
            this._privateKey = n;
```

```
        }

        private BigInteger _privateKey;

        public byte[] ByteArray => _privateKey.ToByteArray();
        public BigInteger BigInteger => _privateKey;
    }

    public class PublicKey
    {
        public PublicKey(Punto p)
        {
            this._publicKey = p;
        }

        private Punto _publicKey;

        public byte[] ByteArray =>
            _publicKey.GetUncompressedByteArray();
        public Punto Point => this._publicKey;
    }
}
```

B.1.2. Clase “FirmaElectronica”

```
using ASN1_Generator;
using System.Numerics;

namespace LibreriaCriptografica
{
    public static class FirmaElectronica
    {
        public static byte[] GenerarFirmaElectronica(byte[] message,
            BigInteger privateKey)
        {
            while (true)
            {
                //Genero un valor 'k' aleatorio
                BigInteger k = Helper.Random(Params.n);

                //Calculo el punto P = k*G
                Punto kP = k * Params.G;

                BigInteger r = Helper.Mod(kP.x, Params.n);

                if (r.IsZero) continue;

                var eHash = FirmaElectronica.ObtenerHash(message);
                var e = new BigInteger(eHash);
                if (e.Sign == -1) e = BigInteger.Negate(e);

                BigInteger s = Helper.Mod(
                    Helper.Inversion_ExtendedEuclidean(k, Params.n) *
                        (e + privateKey * r), Params.n
                    );
```

```
            var asn1 = new ASN1_Generator.ASN1_Generator();
            var seq = new ASN1_Generator.ASN1_Sequence();

            seq.content.Add(new ASN1_Generator.ASN1_Integer() {
                Value = r });
            seq.content.Add(new ASN1_Generator.ASN1_Integer() {
                Value = s });

            asn1.Elements.Add(seq);

            return asn1.GetBytes();
        }
    }

    public static bool VerificarFirmaElectronica(Punto publicKey,
        byte[] message, byte[] signature)
    {
        var parsed = ASN1_Generator.ASN1_Generator.Parse(signature);

        var r = ((ASN1_Integer)((ASN1_Sequence)parsed
            .Elements[0])
            .content[0])
            .Value;

        var s = ((ASN1_Integer)((ASN1_Sequence)parsed
            .Elements[0])
            .content[1])
            .Value;

        if (!(0 < r && r < Params.n)) return false;
        if (!(0 < s && s < Params.n)) return false;

        var eHash = FirmaElectronica.ObtenerHash(message);

        var e = new BigInteger(eHash);
        if (e.Sign == -1) e = BigInteger.Negate(e);

        var w = Helper.Inversion_ExtendedEuclidean(s, Params.n);

        var u1 = Helper.Mod(e * w, Params.n);

        var u2 = Helper.Mod(r * w, Params.n);

        var X = u1 * Params.G + u2 * publicKey;

        if (X.EsInfinito || X.EsCero) return false;

        var v = Helper.Mod(X.x, Params.n);

        if (v == r) return true;
        else return false;
    }

    public static byte[] ObtenerHash(byte[] message)
    {
        var sha256 = System.Security.Cryptography.SHA256.Create();
        var eHash = sha256.ComputeHash(message);
        return eHash;
    }
```

```
        }
}
```

B.1.3. Clase "CertificateSignatureRequest"

Para la generación de Solicitudes de Firmado de Certificado (Certificate Signature Request, en inglés) fue empleada la libería *Bouncy Castle para C#* [1]. Mediante esta librería son generados las Solicitudes de Firmado de Certificado empleando la especificación PKCS#10[2], una especificación prácticamente universal para este tipo de documentos.

```
using Org.BouncyCastle.Asn1;
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Asn1.X509;
using Org.BouncyCastle.Asn1.X9;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Pkcs;
using Org.BouncyCastle.Security;
using System;
using System.Collections.Generic;
using System.Linq;

namespace LibreriaCriptografica
{
    public class CertificateSigningRequest
    {
        public static string GeneratePkcs10
            (PrivateKey privKey, PublicKey pubKey, string commonName,
               string organization, string organizationUnit, string
               city, string state,
             string countryIso2Characters, string email)
        {
            try
            {
                Org.BouncyCastle.Math.BigInteger d = new
                   Org.BouncyCastle.Math.BigInteger(
                        privKey.ByteArray.Reverse().ToArray()
                );

                SecureRandom secureRandom = new SecureRandom();
                X9ECParameters curve =
                   SecNamedCurves.GetByName("secp256k1");
                ECDomainParameters domain = new
                   ECDomainParameters(curve.Curve, curve.G, curve.N,
                   curve.H);

                Org.BouncyCastle.Math.EC.ECPoint q =
                   domain.G.Multiply(d);

                var priv = new ECPrivateKeyParameters("EC", d,
                   SecObjectIdentifiers.SecP256k1);
                var pub = new ECPublicKeyParameters("EC",q,
                   SecObjectIdentifiers.SecP256k1);

                Dictionary <DerObjectIdentifier, string> attrs = new
                   Dictionary<DerObjectIdentifier, string>();

                attrs.Add(X509Name.O, organization);
```

[1] http://www.bouncycastle.org/csharp/

[2] Magnus Nystrom y Burt Kaliski. 2000. *PKCS #10: Certification Request Syntax Specification Version 1.7*. RFC 2986. DOI: 10.17487/RFC2986.

```
                attrs.Add(X509Name.OU, organizationUnit);
                attrs.Add(X509Name.EmailAddress, email);
                attrs.Add(X509Name.L, city);
                attrs.Add(X509Name.ST, state);
                attrs.Add(X509Name.C, countryIso2Characters);
                attrs.Add(X509Name.CN, commonName);

                var subject = new X509Name(attrs.Keys.ToList(),
                    attrs.Values.ToList());

                var pkcs10CertificationRequest = new
                    Pkcs10CertificationRequest("SHA1withECDSA", subject,
                    pub, null, priv);
                var base64csr = Convert.ToBase64String(
                        pkcs10CertificationRequest.GetEncoded()
                        );

                var width = 64;
                for (int i = width; i < base64csr.Length; i += width +
                    1) base64csr = base64csr.Insert(i,"\n");

                return base64csr;
            }
            catch (Exception ex)
            {
                throw;
            }
        }
    }
}
```

B.1.4. Clase “SecurityData”

Esta clase implementa los contenidos detallados en la sección 4.3.6.5.

```
using System;
using System.Linq;
using System.Security.Cryptography;

namespace LibreriaCriptografica
{
    public class SecurityData
    {
        /// <summary>
        /// Tamaño de clave a emplear para encriptación AES (256 bits)
        /// </summary>
        public const int keySize = 256;

        /// <summary>
        /// Tamaño del Vector de Inicialización para AES (IV 128 bits).
           Debe ser del mismo tamaño que el Block Size de AES, para el
           modo CBC.
        /// </summary>
        public const int ivSize = 128;

        /// <summary>
        /// Tamaño del Salt para la derivación del password.
        /// </summary>
        public const int saltSize = 128;

        /// <summary>
        /// Método para crear un nuevo conjunto de datos de seguridad,
           empleando la password provista.
        /// </summary>
        /// <param name="password"></param>
        /// <returns>Nuevo SecurityData</returns>
        public static SecurityData CrearNuevo(string password)
        {
            using (var rand = RandomNumberGenerator.Create())
            {
                byte[] k1 = new byte[keySize / 8];
                rand.GetBytes(k1);
                Console.WriteLine("k1 = " + Convert.ToBase64String(k1));
                return GenerateValues(k1, password);
            }
        }


        private static SecurityData GenerateValues(byte[] k1, string
           password)
        {

            var rand = RandomNumberGenerator.Create();

            byte[] ivBytes = new byte[ivSize / 8];
            rand.GetBytes(ivBytes);

            byte[] saltBytes = new byte[saltSize / 8];
            rand.GetBytes(saltBytes);
```

```
            Rfc2898DeriveBytes rfc2898DeriveBytes = new
                Rfc2898DeriveBytes(password, saltBytes);

            var derived = rfc2898DeriveBytes.GetBytes(64);

            var k2 = new byte[32];
            var k3 = new byte[32];
            Array.Copy(derived, 0, k2, 0, 32);
            Array.Copy(derived, 32, k3, 0, 32);

            var aesEncryption = Aes.Create();
            aesEncryption.KeySize = keySize;
            aesEncryption.BlockSize = 128;
            aesEncryption.Key = k2;
            aesEncryption.IV = ivBytes;
            ICryptoTransform crypto = aesEncryption.CreateEncryptor();
            // The result of the encryption and decryption
            var k1_encrypted = crypto.TransformFinalBlock(k1, 0,
                k1.Length);

            var securityData = new SecurityData()
            {
                k1_encrypted = k1_encrypted,
                k3 = k1,
                salt = saltBytes,
                iv = ivBytes
            };

            return securityData;
        }

        private SecurityData() { }


        public SecurityData(string securityString)
        {
            var splittedSecurityString = securityString.Split('|');
            this.k1_encrypted =
                Convert.FromBase64String(splittedSecurityString[0]);
            this.k3 =
                Convert.FromBase64String(splittedSecurityString[1]);
            this.iv =
                Convert.FromBase64String(splittedSecurityString[2]);
            this.salt =
                Convert.FromBase64String(splittedSecurityString[3]);
        }

        public string SecurityString =>
            Convert.ToBase64String(this.k1_encrypted) + "|"
                + Convert.ToBase64String(this.k3) + "|"
                + Convert.ToBase64String(this.iv) + "|"
                + Convert.ToBase64String(this.salt);


        public byte[] k1_encrypted;

        public byte[] k3;

        public byte[] iv;
```

```
        public byte[] salt;


        public byte[] Encrypt(byte[] plainData, string password, byte[]
            iv)
        {
            if (iv.Length != (ivSize / 8)) throw new
                ArgumentException("IV debería ser de 128 bits.");

            var k2_retrieved = new byte[keySize / 8];
            var k3_retrieved = new byte[keySize / 8];
            var k1 = new byte[keySize / 8];

            using (Rfc2898DeriveBytes rfc2898DeriveBytes = new
                Rfc2898DeriveBytes(password, this.salt))
            {
                var derived = rfc2898DeriveBytes.GetBytes(keySize * 2 /
                    8);
                Array.Copy(derived, 0, k2_retrieved, 0, keySize / 8);
                Array.Copy(derived, keySize / 8, k3_retrieved, 0,
                    keySize / 8);
            }

            //Comparamos las secuencias K3, para corroborar que la
                contraseña ingresada es correcta.
            if (this.k3.Count() != k3_retrieved.Count() &&
                this.k3.Intersect(k3_retrieved).Count() !=
                this.k3.Count()) throw new Exception();

            using (var aesEncryption = Aes.Create())
            {
                aesEncryption.KeySize = SecurityData.keySize;
                aesEncryption.BlockSize = 128;
                aesEncryption.Key = k2_retrieved;
                aesEncryption.IV = this.iv;
                ICryptoTransform crypto =
                    aesEncryption.CreateDecryptor();
                // The result of the encryption and decryption
                k1 = crypto.TransformFinalBlock(this.k1_encrypted, 0,
                    this.k1_encrypted.Length);
            }

            using (var aesEncryption_forText = Aes.Create())
            {
                aesEncryption_forText.KeySize = SecurityData.keySize;
                aesEncryption_forText.BlockSize = 128;
                aesEncryption_forText.Key = k1;
                aesEncryption_forText.IV = iv;
                ICryptoTransform crypto_forText =
                    aesEncryption_forText.CreateEncryptor();
                //El resultado de la encriptación
                var encryptedText =
                    crypto_forText.TransformFinalBlock(plainData, 0,
                    plainData.Length);

                return encryptedText;
            }
        }
```

```
        public byte[] Decrypt(byte[] encryptedData, string password,
            byte[] iv)
        {
            if (iv.Length != (ivSize / 8)) throw new
                ArgumentException("IV debería ser de 128 bits.");

            var k2_retrieved = new byte[keySize / 8];
            var k3_retrieved = new byte[keySize / 8];
            var k1 = new byte[keySize / 8];

            using (Rfc2898DeriveBytes rfc2898DeriveBytes = new
                Rfc2898DeriveBytes(password, this.salt))
            {
                var derived = rfc2898DeriveBytes.GetBytes(keySize * 2 /
                    8);
                Array.Copy(derived, 0, k2_retrieved, 0, keySize / 8);
                Array.Copy(derived, keySize / 8, k3_retrieved, 0,
                    keySize / 8);
            }

            //Comparamos las secuencias K3, para corroborar que la
                contraseña ingresada es correcta.
            if (this.k3.Count() != k3_retrieved.Count() &&
                this.k3.Intersect(k3_retrieved).Count() !=
                this.k3.Count()) throw new Exception();

            using (var aesEncryption = Aes.Create())
            {
                aesEncryption.KeySize = SecurityData.keySize;
                aesEncryption.BlockSize = 128;

                aesEncryption.Key = k2_retrieved;
                aesEncryption.IV = this.iv;
                ICryptoTransform crypto =
                    aesEncryption.CreateDecryptor();
                //Resultado de desencriptación
                k1 = crypto.TransformFinalBlock(this.k1_encrypted, 0,
                    this.k1_encrypted.Length);
            }

            using (var aesEncryption_forText = Aes.Create())
            {

                aesEncryption_forText.KeySize = SecurityData.keySize;
                aesEncryption_forText.BlockSize = 128;
                aesEncryption_forText.Key = k1;
                aesEncryption_forText.IV = iv;
                ICryptoTransform crypto_forText =
                    aesEncryption_forText.CreateDecryptor();
                //Resultado de desencriptación
                var decryptedText =
                    crypto_forText.TransformFinalBlock(encryptedData, 0,
                    encryptedData.Length);
                //Console.WriteLine("decryptedData = " +
                    Convert.ToBase64String(encryptedText));

                return decryptedText;
            }
```

```
        }

        public void PasswordChange(string oldPassword, string
            newPassword)
        {
            var k2_retrieved = new byte[keySize / 8];
            var k3_retrieved = new byte[keySize / 8];
            var k1 = new byte[keySize / 8];

            using (Rfc2898DeriveBytes rfc2898DeriveBytes = new
                Rfc2898DeriveBytes(oldPassword, this.salt))
            {
                var derived = rfc2898DeriveBytes.GetBytes(keySize * 2 /
                    8);
                Array.Copy(derived, 0, k2_retrieved, 0, keySize / 8);
                Array.Copy(derived, keySize / 8, k3_retrieved, 0,
                    keySize / 8);
            }

            //Comparamos las secuencias K3, para corroborar que la
                contraseña ingresada es correcta.
            if (this.k3.Count() != k3_retrieved.Count() &&
                this.k3.Intersect(k3_retrieved).Count() !=
                this.k3.Count()) throw new Exception();

            using (var aesEncryption = Aes.Create())
            {
                aesEncryption.KeySize = SecurityData.keySize;
                aesEncryption.BlockSize = 128;
                aesEncryption.Key = k2_retrieved;
                aesEncryption.IV = this.iv;
                ICryptoTransform crypto =
                    aesEncryption.CreateDecryptor();

                //Resultado de la desencriptación
                k1 = crypto.TransformFinalBlock(this.k1_encrypted, 0,
                    this.k1_encrypted.Length);
            }

            var newSecurity = GenerateValues(k1, newPassword);

            this.k1_encrypted = newSecurity.k1_encrypted;
            this.k3 = newSecurity.k3;
            this.iv = newSecurity.iv;
            this.salt = newSecurity.salt;
        }
    }
}
```

B.2. Sub-librería de Operaciones Aritméticas

B.2.1. Clase “Punto”

La clase “Punto” es tal vez la más importante en cuanto a complejidad y alcance dentro de la Librería Criptográfica. Todo el funcionamiento del Esquema Criptográfico de Curvas Elípticas se basa en operaciones aritméticas sobre puntos de una curva elíptica, tal como fué detallado

en el la sección 2.3.2. En esta sección podremos observar los diferentes elementos que fueron implementados de este componente.

B.2.1.1. Clase parcial "Punto": Definición de propiedades

```
using System;
using System.Linq;
using System.Numerics;

namespace LibreriaCriptografica
{
    public partial class Punto
    {
        public static Punto Infinito
        {
            get
            {
                return new Punto()
                {
                    _EsInfinito = true,
                    _x = BigInteger.Zero,
                    _y = BigInteger.Zero
                };
            }
        }

        public Punto Negar {
            get {
                return new Punto() { X = this.X, Y =
                   Helper.Mod(-this.Y, Params.p), Z = this.Z};
            }
        }

        public Punto(BigInteger x, BigInteger y)
        {
            this.X = x;
            this.Y = y;
            this.Z = 1;
        }

        public Punto(byte[] byteArray)
        {

            if (byteArray.Last() == 0x04)
            {
                var coordByteLength = (byteArray.Length - 1) / 2;

                byte[] xBytes = new byte[coordByteLength];

                byte[] yBytes = new byte[coordByteLength];

                Array.Copy(byteArray, coordByteLength, xBytes, 0,
                   coordByteLength);
                Array.Copy(byteArray, 0, yBytes, 0, coordByteLength);

                var X =
                   BigInteger_Extensions.CreateUnsignedBigInteger(xBytes);
```

```
                var Y =
                    BigInteger_Extensions.CreateUnsignedBigInteger(yBytes);

                if (X.Sign == -1) X = BigInteger.Negate(X);
                if (Y.Sign == -1) Y = BigInteger.Negate(Y);

                this.X = X;
                this.Y = Y;
                this.Z = 1;
            }
        }

        public Punto()
        {
        }

        private bool _EsInfinito;
        public bool EsInfinito { get { return _EsInfinito; } }
        public bool EsCero { get { if (X == 0 && X == 0 && Y == 0)
            return true; else return false; } }

        private BigInteger _x;
        private BigInteger _y;

        public BigInteger x
        {
            get
            {
                if (this.EsInfinito) throw new Exception();

                //return _x;

                return Helper.Mod(X *
                    Helper.Inversion_ExtendedEuclidean(BigInteger.Pow(Z,
                    2), Params.p), Params.p);
            }
            set
            {
                this._x = value;
            }
        }

        public BigInteger y
        {
            get
            {
                if (this.EsInfinito) throw new Exception();
                return Helper.Mod(Y *
                    Helper.Inversion_ExtendedEuclidean(BigInteger.Pow(Z,
                    3), Params.p), Params.p);
            }
            set
            {
                this._y = value;
            }
        }

        public BigInteger X;
        public BigInteger Y;
```

```
        public BigInteger Z;

        public byte[] GetCompressedByteArray() {
            var xByteArray = x.ToByteArray();
            var result = new byte[xByteArray.Length + 1];
            xByteArray.CopyTo(result, 0);
            result[result.Length - 1] = 0x02;
            return result;
        }

        public byte[] GetUncompressedByteArray()
        {
            var xByteArray = RemoveTrailingZeros(x.ToByteArray());
            var yByteArray = RemoveTrailingZeros(y.ToByteArray());
            var result = new byte[xByteArray.Length + yByteArray.Length
               + 1];
            yByteArray.CopyTo(result, 0);
            xByteArray.CopyTo(result, yByteArray.Length);
            result[result.Length - 1] = 0x04;
            return result;
        }

        public static byte[] RemoveTrailingZeros(byte[] data) {

            byte[] foo = data;
            // populate foo
            int i = foo.Length - 1;
            while (foo[i] == 0)
                --i;
            // now foo[i] is the last non-zero byte
            byte[] bar = new byte[i + 1];
            Array.Copy(foo, bar, i + 1);

            return bar;
        }
    }
}
```

B.2.1.2. Clase parcial "Punto": Definición de operaciones

```
using System.Numerics;

namespace LibreriaCriptografica
{
    partial class Punto
    {

        public static Punto operator *(Punto P, BigInteger k) => k*P;
        public static Punto operator *(BigInteger k, Punto P) {
            if (k == 2) {
                return P + P;
            }
            return BinaryNAFPointMultiplication(k, P);
        }

        public static Punto operator +(Punto P, Punto Q) {
```

```
            if (P.X == Q.X && P.Y == Q.Y && P.Z == Q.Z) {
                return Jacobian_PointDoubling(P);
            }
            else if (P.Z == Q.Z) return Zadd_2007_m(P, Q);
            else {
                OperationCounter.Instance.Sums++;
                return Bernstein_Lange_11M_5S_Addition(P, Q);
            }
        }

        public static Punto operator -(Punto P, Punto Q)
        {
            return P + Q.Negar;
        }
    }
}
```

B.2.1.3. Clase parcial "Punto": Implementaciones de algoritmos de suma

```
using System;
using System.Numerics;

namespace LibreriaCriptografica
{
    partial class Punto
    {

        private static Punto GenericAddition(Punto P, Punto Q) {

            if (P.EsCero && !Q.EsCero) return Q;
            if (!P.EsCero && Q.EsCero) return P;
            if (P.EsCero && Q.EsCero) return P;

            if (P.EsInfinito && !Q.EsInfinito) return Q;
            if (!P.EsInfinito && Q.EsInfinito) return P;

            if (P.EsInfinito && Q.EsInfinito) return Punto.Infinito;
            if (P.y == BigInteger.Negate(Q.y)) return Punto.Infinito;

            if (P.x.Sign == -1 || P.y.Sign == -1 || Q.x.Sign == -1 ||
               Q.y.Sign == -1) throw new Exception();

            BigInteger m;

            try
            {
                if (P.x == Q.x && P.y == Q.y) m = Helper.Mod((3 *
                   BigInteger.Pow(P.x, 2) + Params.a) *
                   Helper.Inversion_ExtendedEuclidean(2 * P.y,
                   Params.p), Params.p);
                else m = Helper.Mod(((P.y - Q.y) *
                   Helper.Inversion_ExtendedEuclidean(P.x - Q.x,
                   Params.p)), Params.p);
            }
            catch (Helper.ZeroMultiplicativeInverseException e)
```

```
            {
                return new Punto(0, 0);
            }

            BigInteger xr = Helper.Mod(BigInteger.Pow(m, 2) - P.x -
                Q.x, Params.p);

            BigInteger yr = Helper.Mod(-P.y + m * (P.x - xr), Params.p);

            return new Punto(xr, yr);
        }



        private static Punto Zadd_2007_m(Punto P, Punto Q) {
            BigInteger A, B, C, D, X3, Y3, Z3;

            A = BigInteger.Pow(Q.X - P.X, 2);
            B = P.X * A;
            C = Q.X * A;
            D = BigInteger.Pow(Q.Y - P.Y,2);
            X3 = D - B - C;
            Y3 = (Q.Y - P.Y) * (B - X3) - P.Y * (C - B);
            Z3 = P.Z * (Q.X - P.X);

            return new Punto() { X = Helper.Mod(X3, Params.p), Y =
                Helper.Mod(Y3, Params.p), Z = Helper.Mod(Z3, Params.p) };
        }

        private static Punto Bernstein_Lange_11M_5S_Addition(Punto P,
            Punto Q) {
            if (Q.EsInfinito || Q.EsCero)
                return P;
            if (P.EsInfinito || P.EsCero)
                return Q;
            if ((P.EsInfinito && Q.EsInfinito))
                return Punto.Infinito;

            if (P.X == Q.X && P.Y == Q.Y && P.Z == Q.Z)
                return dbl_2005_dl(P);
            BigInteger Z1Z1, Z2Z2, U1, U2, S1, S2, H, I, J, r, V, X3,
                Y3, Z3;
            Z1Z1 = P.Z * P.Z;
            Z2Z2 = Q.Z * Q.Z;
            U1 = P.X * Z2Z2;
            U2 = Q.X * Z1Z1;
            S1 = P.Y * Q.Z * Z2Z2;
            S2 = Q.Y * P.Z * Z1Z1;
            H = U2 - U1;
            I = BigInteger.Pow(2 * H, 2);
            J = H * I;
            r = 2 * (S2 - S1);
            V = U1 * I;
            X3 = BigInteger.Pow(r, 2) - J - 2 * V;
            Y3 = r * (V - X3) - 2 * S1 * J;
            Z3 = (BigInteger.Pow(P.Z + Q.Z, 2) - Z1Z1 - Z2Z2) * H;
            return new Punto() { X = Helper.Mod(X3,Params.p), Y =
                Helper.Mod(Y3,Params.p), Z = Helper.Mod(Z3,Params.p) };
        }
```

```
        private static Punto AffineJacobian_PointAddition(Punto P,
            Punto Q) {

            if (Q.EsInfinito || Q.EsCero)
                return P;
            if (P.EsInfinito || P.EsCero)
                return Q;
            if ((P.EsInfinito && Q.EsInfinito))
                return Punto.Infinito;

            BigInteger  X1 = P.X,
                        Y1 = P.Y,
                        Z1 = P.Z,
                        X2 = Q.x,
                        Y2 = Q.y,
                        Z2 = 1;

            BigInteger T1 = (Y2 * BigInteger.Pow(Z1, 3) - Y1);
            BigInteger T2 = X2 * BigInteger.Pow(Z1, 2) - X1;
            BigInteger T3 = BigInteger.Pow(T2, 2);
            BigInteger X3 = Helper.Mod(BigInteger.Pow(T1,2) - T3 * (X1
                + X2*BigInteger.Pow(Z1,2)),Params.p);
            BigInteger Y3 = Helper.Mod(T1 * (X1 * T3 - X3) - Y1 *
                BigInteger.Pow(X2 * BigInteger.Pow(Z1, 2) - X1,
                3),Params.p);
            BigInteger Z3 = Helper.Mod((T2) * Z1,Params.p);
            return new Punto() { X = X3, Y = Y3, Z = Z3};
        }
    }
}
```

B.2.1.4. Clase parcial “Punto”: Implementaciones de algoritmos de duplicación

```
using System.Numerics;

namespace LibreriaCriptografica
{
    partial class Punto
    {
        public static Punto Jacobian_PointDoubling(Punto P)
        {
            if (P.EsInfinito) return Punto.Infinito;
            BigInteger T1 = 3 * BigInteger.Pow(P.X, 2) + Params.a *
                BigInteger.Pow(P.Z, 4);
            BigInteger T2 = P.X * BigInteger.Pow(P.Y, 2);
            BigInteger T3 = 4 * T2;
            BigInteger X3 = Helper.Mod(BigInteger.Pow(T1, 2) - 2 * T3,
                Params.p);
            BigInteger Y3 = Helper.Mod((T1) * (T3 - X3) - 8 *
                BigInteger.Pow(P.Y, 4), Params.p);
            BigInteger Z3 = Helper.Mod(2 * P.Y * P.Z, Params.p);
            return new Punto() { X = X3, Y = Y3, Z = Z3 };
        }
```

```
        public static Punto dbl_2005_dl(Punto P)
        {
            if (P.EsInfinito) return Punto.Infinito;
            BigInteger XX = 0, YY = 0, YYYY, ZZ, S, M, T, X3, Y3, Z3;
            XX = BigInteger.Pow(P.X, 2);
            YY = BigInteger.Pow(P.Y, 2);
            YYYY = BigInteger.Pow(YY, 2);
            ZZ = BigInteger.Pow(P.Z, 2);
            S = 2 * (BigInteger.Pow(P.X + YY, 2) - XX - YYYY);
            M = 3 * XX + Params.a * BigInteger.Pow(ZZ, 2);
            T = BigInteger.Pow(M, 2) - 2 * S;
            X3 = T;
            Y3 = M * (S - T) - 8 * YYYY;
            Z3 = BigInteger.Pow(P.Y + P.Z, 2) - YY - ZZ;
            return new Punto() { X = Helper.Mod(X3, Params.p), Y =
                Helper.Mod(Y3, Params.p), Z = Helper.Mod(Z3, Params.p) };
        }
    }
}
```

B.2.1.5. Clase parcial “Punto”: Implementaciones de algoritmos de multiplicación

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;

namespace LibreriaCriptografica
{
    partial class Punto
    {
        public static Punto IterativeMultiplication(BigInteger k, Punto
            P)
        {
            Punto Q = Punto.Infinito;
            for (int i = 1; i <= k; i++)
            {
                Q += P;
            }
            return Q;
        }

        /// <summary>
        /// Algorithm 3.30: Computing the NAF of a positive integer
        /// </summary>
        /// <param name="k"></param>
        /// <returns></returns>
        public static List<BigInteger> computeNAF(BigInteger k)
        {
            List<BigInteger> NAF = new List<BigInteger>();
            while (k >= 1)
            {
                BigInteger ki = 0;
                if (!k.IsEven)
                {
```

```
                ki = 2 - (Helper.Mod(k, 4));
                k -= ki;
            }
            NAF.Add(ki);
            k /= 2;
        }
        return NAF;
    }

    /// <summary>
    /// Algorithm 3.31: BInary NAF method for point multiplication
    /// </summary>
    /// <param name="k"></param>
    /// <param name="P"></param>
    /// <returns></returns>
    public static Punto BinaryNAFPointMultiplication(BigInteger k,
        Punto P)
    {
        var NAF = computeNAF(k);
        Punto Q = Punto.Infinito;
        NAF.Reverse();
        foreach (BigInteger ki in NAF)
        {
            Q = 2*Q;

            if (ki == 1) Q += P;
            else if (ki == -1) Q -= P;
        }
        return Q;
    }

    /// <summary>
    /// Algorithm 3.27: Left-to-right binary method for point
        multiplication
    /// </summary>
    /// <param name="k"></param>
    /// <param name="P"></param>
    /// <returns></returns>
    public static Punto LeftToRight_BinaryMultiplication(BigInteger
        k, Punto P) {
        string kBin = k.ToBinaryString();
        Punto Q = Punto.Infinito;
        foreach (char ki in kBin) {
            Q = 2 * Q;
            if (ki == '1') Q = Q + P;
        }
        return Q;
    }

    /// <summary>
    /// Algorithm 3.26: Right-to-left binary method for point
        multiplication
    /// </summary>
    /// <param name="k"></param>
    /// <param name="P"></param>
    /// <returns></returns>
    public static Punto RightToLeft_BinaryMultiplication(BigInteger
        k, Punto P)
    {
```

```
            string kBin = k.ToBinaryString();
            var rightToLeft = kBin.Reverse();
            Punto Q = Punto.Infinito;
            foreach (char ki in rightToLeft)
            {
                if (ki == '1') Q = Q + P;
                P = 2 * P;
            }
            return Q;
        }

        /// <summary>
        /// Algorithm 3.35: Computing the width-w NAF of a positive
            integer
        /// </summary>
        /// <param name="k"></param>
        /// <param name="w"></param>
        /// <returns></returns>
        public static List<BigInteger> compute_wNAF(BigInteger k, int w)
        {
            if (k < 2) throw new Exception("You dumb fuck");

            List<BigInteger> NAF = new List<BigInteger>();

            while (k >= 1)
            {
                if (!k.IsEven)
                {
                    BigInteger ki = mods(k, w);
                    k -= ki;
                    NAF.Insert(0, ki);
                }
                else {
                    BigInteger ki = 0;
                    NAF.Insert(0, ki);
                }

                k /= 2;

            }

            return NAF;
        }

        /// <summary>
        /// Algorithm 3.36: Window NAF method for point multiplication
        /// </summary>
        /// <param name="k"></param>
        /// <param name="P"></param>
        /// <param name="w"></param>
        /// <returns></returns>
        public static Punto WindowNAFPointMultiplication(BigInteger k,
            Punto P)
        {

            var wNAF = compute_wNAF(k,
                WindowNAFPointMultiplication_Precompute.Instance.w);

            Punto Q = Punto.Infinito;
```

```
            foreach (BigInteger ki in wNAF)
            {
                Q = 2 * Q;

                if (ki != 0) {
                    if(ki > 0) Q +=
                        WindowNAFPointMultiplication_Precompute

                    else Q -= WindowNAFPointMultiplication_Precompute

                }
            }

            return Q;
        }

        public static BigInteger mods(BigInteger k, int window)
        {
            OperationCounter.Instance.Mods++;
            BigInteger A_2powWindow = BigInteger.Pow(2, window);
            BigInteger remainder = BigInteger.Remainder(k,
                A_2powWindow);
            if (remainder >= BigInteger.Pow(2, window - 1))
                return (remainder) - A_2powWindow;
            else return (remainder);
        }
    }
}
```

B.2.2. Clase “Params”

```
using System.Numerics;

namespace LibreriaCriptografica
{
    public class Params {
        public static BigInteger p =
            Helper.ParseHexToPositiveBigInteger("FFFFFFFF FFFFFFFF
            FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F");
        public static BigInteger a =
            Helper.ParseHexToPositiveBigInteger("00000000 00000000
            00000000 00000000 00000000 00000000 00000000 00000000");
        public static BigInteger b =
            Helper.ParseHexToPositiveBigInteger("00000000 00000000
            00000000 00000000 00000000 00000000 00000000 00000007");

        public static BigInteger n =
            Helper.ParseHexToPositiveBigInteger("FFFFFFFF FFFFFFFF
            FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141");

        public static Punto G = new
            Punto(Helper.ParseHexToPositiveBigInteger("79BE667E F9DCBBAC
            55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798"),
             Helper.ParseHexToPositiveBigInteger("483ADA77 26A3C465
                5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8")
```

```
            );

        public static int bits = 256;
    }
}
```

B.2.3. Clase “Aritmetica”

```
using System;
using System.Numerics;

namespace LibreriaCriptografica
{
    class Arithmetic
    {
        public static BigInteger Mods(BigInteger k, int window)
        {
            BigInteger A_2powWindow = BigInteger.Pow(2, window);
            BigInteger remainder = BigInteger.Remainder(k,
                A_2powWindow);
            if (remainder >= BigInteger.Pow(2, window - 1))
                return (remainder) - A_2powWindow;
            else return (remainder);
        }

        public static BigInteger Mod(BigInteger n, BigInteger p)
        {
            BigInteger r = n % p;
            if (((p > 0) && (r < 0)) || ((p < 0) && (r > 0))) r += p;
            return r;
        }

        public static BigInteger Inversion_ExtendedEuclidean(BigInteger
            a, BigInteger p)
        {

            if (p < 0) p = -p;
            if (a < 0) a = p - (Arithmetic.Mod(-a, p));

            if (a == 0) throw new ZeroMultiplicativeInverseException();
            BigInteger u = a, v = p;
            BigInteger q, r, x;
            BigInteger x1 = 1, x2 = 0;


            while (u != 1)
            {
                q = DivideAndFloor(v, u);
                r = v - q * u;
                x = x2 - q * x1;
                v = u;
                u = r;
                x2 = x1;
                x1 = x;
            }
            return Arithmetic.Mod(x1, p);
```

```
        }

        public static BigInteger DivideAndFloor(BigInteger a,
            BigInteger b)
        {
            BigInteger result = a / b;
            BigInteger resto = BigInteger.Remainder(a, b);
            if (result.Sign == -1 && resto != 0) return result - 1;
            else return result;
        }

        public class ZeroMultiplicativeInverseException : Exception
        {
        }
    }
}
```

B.2.4. Clase "Helper"

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.Numerics;
using System.Text.RegularExpressions;

namespace LibreriaCriptografica
{
    public static class Helper
    {
        public static BigInteger Random(BigInteger n)
        {

            Random random = new System.Random();
            var randomData = n.ToByteArray();

            BigInteger max = BigInteger.Pow(256, randomData.Length) - 1;

            random.NextBytes(randomData);

            byte[] unsignedRandomData = new byte[randomData.Length + 1];
            Array.Copy(randomData, unsignedRandomData,
                randomData.Length);
            unsignedRandomData[unsignedRandomData.Length - 1] = 0x0;

            var randomBigInteger = new BigInteger(unsignedRandomData);

            return randomBigInteger * n / max;

        }

        public static BigInteger ParseHexToPositiveBigInteger(String
            str)
        {
            return BigInteger.Parse("00000000" + Regex.Replace(str,
                @"\s+", ""), NumberStyles.AllowHexSpecifier); ;
```

```
            }
        }
    }
```