



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Trabajo Terminal II.

**Autentificación mediante Chaffing and
Winnowing en el protocolo HTTP**

TT2018-B003.

Integrantes:

Blancas Pérez Bryan Israel
Carrillo Fernández Gerardo
Morales González Diego Arturo
Paredes Hernández Pedro Antonio

Directores:

Moreno Cervantes Axel Ernesto
Díaz Santiago Sandra

Índice.

1. Introducción	11
1.1. Objetivos	12
1.1.1. Objetivo general	12
1.1.2. Objetivos particulares	12
1.2. Trabajo previo	13
1.2.1. Kerberos (CISCO)	13
1.2.2. Tririga (IBM)	13
1.3. Justificación	15
1.3.1. Limites y alcances	16
1.4. Metodología	17
1.5. Organización del documento	20
1.5.1. Capítulo 2. Marco teórico.	20
1.5.2. Capítulo 3. Análisis.	20
1.5.3. Capítulo 4. Diseño.	20
1.5.4. Capítulo 5. Desarrollo.	20
1.5.5. Capítulo 6. Avances y trabajo por hacer.	21
2. Marco teórico	22
2.1. Protocolo HTTP	22
2.1.1. Solicitud HTTP	23
2.1.2. Respuesta HTTP	23
2.2. Certificado	24
2.2.1. Autoridad Certificadora.	25
2.3. Métodos de autentificación en internet.	26
2.4. Cookies.	29
2.5. Introducción a la Criptografía.	31
2.5.1. Criptología.	31
2.5.2. Criptografía.	31
2.5.3. Objetivos de la criptografía.	33
2.5.4. Usos de la criptografía.	33
2.5.5. Criptografía simétrica y asimétrica.	34

2.5.6. Criptografía a nivel de aplicación.	37
2.6. Chaffing and Winnowing.	39
2.6.1. Historia	39
2.6.2. ¿Qué es Chaffing and Winnowing?	39
2.6.3. Objetivo de Chaffing and Winnowing	41
2.6.4. ¿Cómo funciona?	42
2.6.5. Propiedades de Chaffing and Winnowing	43
2.6.6. All-or-Nothing and the Package Transform (AONT) .	44
2.6.7. Comparando Chaffing and Winnowing contra Cifrado y Esteganografía	46
Prototipo 1.	49
3. Análisis.	50
3.1. Estudio de Factibilidad.	50
3.1.1. Factibilidad Técnica	50
3.1.2. Factibilidad Operativa	52
3.1.3. Factibilidad Económica	53
3.2. Herramientas a usar.	54
3.2.1. Software.	54
3.2.2. Hardware.	64
3.3. Arquitectura del sistema.	65
3.3.1. Descripción de la arquitectura del sistema.	65
3.4. Diagrama BPMN	67
3.4.1. Diagrama BPMN Proceso: Inicio de Sesión	68
3.5. Diagrama de casos de uso general.	69
3.6. Componente I. Extensión.	69
3.6.1. Descripción.	69
3.6.2. Estudio de requerimientos.	70
3.6.3. Reglas del negocio.	72
3.7. Componente II: Servidor autentificador.	74
3.7.1. Descripción.	74
3.7.2. Estudio de requerimientos.	74
3.7.3. Reglas del negocio.	75
3.8. Componente III: API.	76
3.8.1. Descripción.	76
3.8.2. Estudio de requerimientos.	77
3.8.3. Reglas del negocio.	78

4. Diseño.	80
4.1. Componente I: Extensión.	80
4.1.1. Diagrama de casos de uso	80
4.1.2. Diagrama de flujo.	93
4.1.3. Diagrama de flujo de datos.	95
4.1.4. Diagrama de clases.	96
4.1.5. Diagramas de secuencia.	99
4.1.6. Diagrama de actividades	106
4.1.7. Interfaz de usuario.	107
4.1.8. Requisitos de diseño.	117
4.1.9. Algoritmos.	118
4.2. Componente II: Servidor Autentificador.	122
4.2.1. Diagrama de casos de uso.	122
4.2.2. Diagrama de flujo.	130
4.2.3. Diagrama de flujo de datos.	131
4.2.4. Diagrama de clases.	132
4.2.5. Diagramas de secuencias.	135
4.2.6. Diagrama de actividades.	139
4.3. Componente III: API.	140
4.3.1. Diagrama de casos de uso.	140
4.3.2. Diagrama de flujo.	145
4.3.3. Diagrama de flujo de datos.	146
4.3.4. Diagrama de clases.	147
4.3.5. Diagramas de secuencias.	150
4.3.6. Diagrama de actividades	153
4.3.7. Interfaz de usuario.	153
4.3.8. Algoritmos.	157
5. Desarrollo.	161
5.1. Componente I. Extensión.	161
5.1.1. Archivo popup.html	163
5.1.2. Archivo background.js	163
5.2. Componente II. Servidor Autentificador.	168
5.2.1. Manejador de paquetes de Node (npm).	168
5.2.2. Componentes.	169
5.3. Componente III. API.	176
5.3.1. API	176
5.3.2. Servicio Web.	181

6. Pruebas.	188
6.1. Pruebas de integración.	188
6.1.1. Extensión y Servidor autentificador.	188
6.1.2. Extensión y Servicio Web.	189
6.1.3. Servidor Autentificador y API.	190
6.2. Tiempos de ejecución.	191
6.2.1. Algoritmo de Chaffing.	191
6.2.2. Algoritmo de Winnowing.	191
6.2.3. Inicio de sesión.	192
Prototipo 2.	193
7. Análisis.	194
7.1. Arquitectura del sistema.	194
7.1.1. Descripción de la arquitectura del sistema.	194
7.2. Componente I. Extensión.	195
7.2.1. Descripción.	195
7.2.2. Cambios planteados.	196
7.2.3. Cambios al estudio de requerimientos y reglas de negocio.196	
7.3. Componente II: Servidor autentificador.	197
7.3.1. Descripción.	197
7.3.2. Cambios Planteados.	197
7.3.3. Cambios al estudio de requerimientos y reglas de negocio.197	
7.4. Componente III: API.	198
7.4.1. Descripción.	198
7.4.2. Cambios planteados.	198
7.4.3. Cambios al estudio de requerimientos y reglas de negocio.199	
8. Diseño.	200
8.1. Componente I: Extensión.	200
8.1.1. Cambios en el algoritmo.	200
8.2. Componente II: Servidor Autentificador.	202
8.2.1. Cambios realizados.	202
8.2.2. Diagrama de Actividades.	202
8.2.3. Diagrama de Secuencia.	203
8.3. Componente III: API.	207
8.3.1. Cambios realizados.	207
8.3.2. Cambio en el algoritmo.	207
8.3.3. Diagrama de Flujo.	209

9. Desarrollo.	211
9.1. Componente I: Extensión.	211
9.1.1. Cambios en el manifest.	211
9.1.2. Implementación del Chaffing.	212
9.2. Componente II: Servidor Autentificador.	214
9.2.1. Servidor de accesos FTP.	214
9.3. Componente III: API.	217
9.3.1. Nuevo Proceso de Winnowing y eliminación del descifrado AES.	217
9.3.2. Guardado del certificado	218
10. Pruebas.	219
10.1. Pruebas de integración.	219
10.1.1. Extensión y Servidor autentificador	219
10.1.2. Extensión y Servicio Web	220
10.1.3. Servidor Autentificador y API	221
10.2. Pruebas funcionales.	222
10.3. Tiempos de ejecución.	224
10.3.1. Algoritmo de Chaffing.	224
10.3.2. Algoritmo de Winnowing.	225
10.3.3. Inicio de sesión.	225
11. Conclusiones	227
12. Trabajo a futuro.	228

Índice de figuras.

1.1. Fases de la metodología por prototipos	18
2.1. Comunicación Cliente - Servidor en el protocolo HTTP	22
2.2. Firma de un certificado por una AC.	25
2.3. Verificación de un certificado firmado por una AC.	25
2.4. Esquema del protocolo de criptografía simétrica.	35
2.5. Esquema del protocolo de criptografía asimétrica.	36
2.6. Charles agrega los paquetes inválidos.	40
2.7. Charles no agrega los paquetes pero multiplexa los flujos.	41
2.8. Secuencia de Chaffing después del proceso de autentificación. .	42
2.9. Formas del proceso de chaff	43
2.10. Visión general del proceso Chaffing and Winnowing.	43
2.11. Proceso de Chaffing and Winnowing junto con AONT.	46
3.1. Arquitectura General del Sistema	65
3.2. Diagrama de Modelo y Notación de Procesos de Negocio . . .	67
3.3. Diagrama BPMN del proceso Iniciar Sesión	68
3.4. Diagrama de casos de uso general del sistema	69
4.1. Diagrama de casos de uso del Componente I.	80
4.2. Diagrama de flujo del Componente I.	93
4.3. Diagrama de flujo de datos del Componente I.	95
4.4. Diagrama de clases de Componente I.	96
4.5. Diagrama de secuencia 1 del Componente I	99
4.6. Diagrama de secuencia 2 del Componente I	100
4.7. Diagrama de secuencia 3 del Componente I	101
4.8. Diagrama de secuencia 4 del Componente I	102
4.9. Diagrama de secuencia 5 del Componente I	103
4.10. Diagrama de secuencia 6 del Componente I	104
4.11. Diagrama de secuencia 7 del Componente I	105
4.12. Diagrama de actividades del Prototipo 2.	106
4.13. Pantalla de interfaz de la extensión.	107

4.14. Pantalla inicial.	108
4.15. Pantalla de aviso.	109
4.16. Tab de la extensión. Permite inicio de sesión	110
4.17. Mensaje de éxito	110
4.18. Mensaje de error	111
4.19. Tab de la extensión. Permite registrarse	112
4.20. Mensaje de éxito	113
4.21. Mensaje de error	113
4.22. Mensaje de error	114
4.23. Mensaje de error	114
4.24. Mensaje de error	115
4.25. Interfaz	115
4.26. Mensaje de éxito	116
4.27. Mensaje de error	116
4.28. Arreglo del patrón de chaffing	120
4.29. Arreglo del patrón de chaffing después de una iteración	120
4.30. Arreglo del patrón de chaffing final	120
4.31. Diagrama de casos de uso del Componente II.	122
4.32. Diagrama de flujo de Componente II.	130
4.33. Diagrama de flujo de datos del Componente II.	131
4.34. Diagrama de clases de Componente II.	132
4.35. Diagrama de secuencia 1 del Componente II	135
4.36. Diagrama de secuencia 2 del Componente II	136
4.37. Diagrama de secuencia 3 del Componente II	137
4.38. Diagrama de secuencia 4 del Componente II	138
4.39. Diagrama de actividades de Componente II.	139
4.40. Diagrama de caso de uso del componente III	140
4.41. Diagrama de flujo de Componente III.	145
4.42. Diagrama de flujo de datos de Componente III.	146
4.43. Diagrama de Clases de componente 3.	147
4.44. Diagrama de Secuencia de Caso de Uso 1. Comprobar certificado.	150
4.45. Diagrama de Secuencia de Caso de Uso 2. Enviar petición. . .	151
4.46. Diagrama de Secuencia de Caso de Uso 3. Redirigir página. . .	152
4.47. Diagrama de Actividades de componente 3.	153
4.48. Mensaje de éxito	154
4.49. Interfaz	155
4.50. Interfaz	155
4.51. Interfaz	156
4.52. Interfaz	156
4.53. Mensaje de error	157

5.1. Estructura del desarrollo de la Extensión.	162
5.2. Estructura del desarrollo de la Autoridad Certificadora.	170
5.3. Estructura del desarrollo de la API.	178
6.1. Sesión iniciada	189
6.2. Resultados de la API al recibir petición de la extensión.	190
6.3. Resultados de la comunicación entre la API y el Servidor Autentificador.	191
7.1. Arquitectura General del Sistema	194
8.1. Ajustes al Diagrama de Actividades del Componente II.	203
8.2. Ajustes al Diagrama de Secuencia crear nuevo usuario del componente II.	204
8.3. Ajustes al Diagrama de Secuencia revocar certificado del componente II.	205
8.4. Ajustes al Diagrama de Secuencia obtener certificado del componente II.	206
8.5. Ajustes al Diagrama de Secuencia verificar certificado del componente II.	206
8.6. Ajustes al Diagrama de Flujo Componente III.	209
10.1. Respuesta de la Autoridad Certificadora hacia la Extensión . .	220
10.2. Sesión iniciada con éxito (Obtención de certificado de la Autoridad certificadora.)	220
10.3. Analizador de protocolos Wireshark de la petición dada por el usuario hacia servicio web.)	221
10.4. Analizador de protocolos Wireshark de la petición dada por la API hacia la autoridad certificadora.)	221
10.5. Inicio de sesión en la extensión	222
10.6. Certificado del cliente devuelto por el Servidor Autentificador	222
10.7. Página de inicio del Servicio Web	223
10.8. Inicio de sesión con la extensión habilitada.	223
10.9. Inicio de sesión exitoso a la cuenta con el certificado asignado.	224

Índice de cuadros.

1.1.	Tabla comparativa de estado del arte	15
2.1.	Aplicación de los métodos de autentificación	27
2.2.	Seguridad en los métodos de autentificación	28
3.1.	Herramientas de Software	51
3.2.	Equipo de Hardware 1	51
3.3.	Equipo de Hardware 2	51
3.4.	Equipo de Hardware 3	52
3.5.	Horas de trabajo	53
4.1.	DCU: CI_CU1	81
4.2.	DCU: CI_CU2	82
4.3.	DCU: CI_CU3	83
4.4.	DCU: CI_CU4	84
4.5.	DCU: CI_CU5	87
4.6.	DCU: CI_CU6	88
4.7.	DCU: CI_CU7	91
4.8.	DCU: CII_CU1	123
4.9.	DCU: CII_CU2	125
4.10.	DCU: CII_CU3	127
4.11.	DCU: CII_CU4	129
4.12.	DCU: CIII_CU1	141
4.13.	DCU: CIII_CU2	143
4.14.	DCU: CIII_CU3	144

Glosario.

Cookies Es una pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del navegador. Sus principales funciones son recordar accesos y conocer información sobre los hábitos de navegación e intentos de spyware. 29

Flash Aplicación informática englobada en la categoría de reproductor multimedia. 55

HTTP Hypertext Transfer Protocol. 70, 81

ID identificador. 26

Identity theft También conocido como "robo de identidad" se produce cuando una persona adquiere, transfiere, posee o utiliza información personal de una persona física o jurídica de forma no autorizada, con la intención de efectuar o vincularlo con algún fraude u otro delito. 28

IU Interfaz de Usuario. 72

Netcape Navegador web de la compañía NetScape Communications. 57

Single Sign-On Es un procedimiento de autenticación que le permite a un usuario determinado acceder a varios sistemas con una sola instancia de identificación.. 14

Capítulo 1

Introducción

Hasta el 2017 la autentificación por contraseña es la más utilizada en los servicios web por su facilidad de implementación, mantenimiento y usabilidad para el usuario. Sin embargo, mientras esta autentificación nos brinda la posibilidad de implementarla fácilmente, no es tan segura como otros métodos y requiere la memorización de las contraseñas [17].

Como consecuencia de ello, los servicios web han implementado mecanismos de seguridad tales como contraseñas con un mínimo de caracteres, al menos un carácter especial, entre otros, lo que ha provocado el surgimiento de herramientas de administración de contraseñas o directamente al almacenado de éstas en medios físicos ó digitales que podrían ser inseguros. Cabe resaltar que las consecuencias al hacer uso de estos medios inseguros pueden ocasionar pérdida de datos sensibles, robo de identidad ó incluso robo de cuentas bancarias [8].

A su vez, Google Chrome ofrece una función conocida como '*Recordar contraseña*', la cual, con el consentimiento del usuario, guarda las credenciales¹ de los servicios web para después mostrarlas o autocompletar el formulario en cuestión. Sin embargo, esas contraseñas se almacenan en archivos de texto que no están cifrados o protegidos, por lo que son vulnerables frente a intrusos. Es decir, cualquiera con acceso al ordenador podría consultar esas contraseñas en la configuración del navegador [5].

Es por ello que en este trabajo terminal queremos brindarle a una aplicación web que cuente con inicio de sesión por contraseña una alternativa de autentificación rápida y segura para que sus usuarios puedan acceder sin la necesidad de utilizar herramientas como las anteriormente mencionadas.

¹Credenciales se entiende como los datos que un servicio web requiere para poder acceder a él. Comúnmente son 'usuario' y 'contraseña'.

1.1. Objetivos.

En esta sección hablaremos de los objetivos tanto general como particulares que se quieren alcanzar al finalizar este trabajo terminal.

1.1.1. Objetivo general.

Realizar un nuevo método de autentificación modificando los datos de la cabecera del protocolo HTTP utilizando Chaffing and Winnowing con la ayuda de una extensión del navegador Google Chrome.

1.1.2. Objetivos particulares.

- Investigar e implementar el desarrollo de extensiones en Google Chrome.
- Investigar sobre los mecanismos de autentificación.
- Investigar sobre la técnica de *Chaffing and Winnowing* para adaptar su implementación.
- Inyectar el certificado autenticador (*Chaffing*) en el encabezado HTTP para enviar la petición al servidor.
- Implementación de un Login en la extensión de Google Chrome.
- Investigar sobre Autoridades Certificadoras para implementar un servidor autenticador.
- Generación de certificado autenticador del usuario
- Implementación de este método de un servicio web de prueba.
- Desarrollo de un API para nuestro servidor que obtenga el certificado del usuario (*Winnowing*).
- Modificar el código del servidor Apache para simular y comprobar el funcionamiento de la extensión.
- Realizar pruebas de seguridad para comprobar la eficacia de la extensión.

1.2. Trabajo previo.

Con la finalidad de darle al lector un panorama más amplio sobre la importancia y el impacto del trabajo terminal a desarrollar, se procede a explicar otros servicios similares al que se presenta en este trabajo:

1.2.1. Kerberos (CISCO)

Kerberos es un servicio confiable de autenticación de terceros basado en el modelo presentado por Needham y Schroeder. Cuando un usuario hace peticiones a un servicio, su identidad debe ser establecida. Para hacer esto, un boleto² se presenta al servidor, junto con la prueba que el boleto fue publicado al usuario, y no fue robado originalmente.

Kerberos guarda en una base de datos sus clientes y sus claves privadas. La clave privada es un número grande que solamente conoce Kerberos y el cliente y la cual esta cifrada. Los servicios de red que requieren la autenticación se registran en Kerberos, al igual que los clientes que desean utilizar esos servicios.

Kerberos también genera las claves privadas temporales, llamadas las claves de la sesión, que se dan a dos clientes y nadie más. Una clave de la sesión se puede utilizar para cifrar los mensajes entre estos.

Cada mensaje no sólo se autentica si no también se cifra. Los mensajes privados son utilizados, por ejemplo, por el servidor de Kerberos para enviar las contraseñas sobre la red [31].

1.2.2. Tririga (IBM)

IBM ha implementado una autenticación de usuarios mediante el inicio de sesión único, la ultima versión (IBM Tririga Application Platform 3.6.0) fue creada en el año 2018 [14]. Esta autenticación está implementada para obtener acceso a **IBM Tririga**³, en esta plataforma es necesario autenticar un usuario como usuario valido del sistema y concederle acceso a las

²Un boleto contiene el nombre del servidor, el nombre del cliente, la dirección de Internet del cliente, un grupo fecha/hora, un curso de la vida, y una clave de sesión aleatoria. Se utiliza para pasar con seguridad la identidad de la persona a quien le fue publicado el boleto entre el servidor de autenticación y el servidor externo.

³IBM Tririga es un sistema de gestión del espacio de trabajo integrado (IWMS) que incrementa el rendimiento operativo, financiero y medioambiental de sus instalaciones y bienes inmuebles.

aplicaciones y funciones de la suite ⁴ de aplicaciones de IBM Tririga. El inicio de sesión único (Single Sign-On) proporciona a los usuarios una vía para gestionar el acceso a varias aplicaciones de su entorno [15].

Tririga Application Platform utiliza su propia autentificación nativa de forma predeterminada. Con la autentificación nativa, el usuario especifica su nombre de usuario y su contraseña en una pantalla de inicio de sesión de IBM Tririga, luego entonces, la plataforma autentifica el usuario comparando el nombre de usuario y la contraseña almacenados en la base de datos de IBM Tririga [15].

El proceso de este tipo de autentificación de Tririga es el siguiente:

- El usuario especifica la URL del servidor web en un navegador o accede a la aplicación mediante un cliente.
- Es posible que se le solicite al usuario que especifique un nombre de usuario o una contraseña, o bien que se inicie la sesión inmediatamente. El inicio de sesión inmediato, sin que el servidor cuestione al navegador o al cliente, no está soportado en algunas configuraciones.
- El servidor web, el servidor de aplicaciones o el plug-in de autenticación verifica la información con el origen de la autenticación.
- Si el inicio de sesión es satisfactorio, el servidor web añade las credenciales de usuario a la cabecera HTTP y las envía al servidor de aplicaciones.
- El servidor de aplicaciones procesa las credenciales de usuario e inicia la sesión en la aplicación.

⁴El término suite hace referencia a un conjunto de aplicaciones y herramientas de software incluidas en un sólo paquete y que, por lo general, comparten un aspecto similar y se integran entre sí.

	Ventajas	Desventajas
Kerberos	<ul style="list-style-type: none"> - Kerberos proporciona una interfaz para los clientes de la aplicación y los servidores de aplicaciones. - Métodos de cifrado seguros y rápidos. - Aislamiento de información sensible y no sensible. - Es posible tener copias de la base de datos en diversas máquinas auxiliares. 	<ul style="list-style-type: none"> - El curso de vida de un boleto debe tener un balance adecuado entre seguridad y conveniencia. - La información puede ser vulnerada si el usuario olvida finalizar su sesión. - Posibles problemas de compatibilidad con las proxys.
Tririga	<ul style="list-style-type: none"> - Soporta los métodos de inserción del nombre de usuario en una cabecera HTTP. - Distinción entre mayúsculas y minúsculas. - Eliminación de nombre de dominio. 	<ul style="list-style-type: none"> - Se puede crear una vulnerabilidad al mantener abierto el puerto HTTP. - Este servicio es exclusivo para un usuario en específico.

Cuadro 1.1: Tabla comparativa de los diferentes servicios

Es por esto que se propone una nueva alternativa de autentificación que brinda al usuario el acceso a cada uno de sus servicios web. El usuario se autentificará una sola vez en cada servicio, sin la necesidad de almacenar sus datos sensibles ⁵ en medios ajenos al servicio.

1.3. Justificación.

Este trabajo terminal busca verificar la viabilidad de implementar Chaffing and Winnowing como una alternativa de autentificación en una aplicación web con inicio de sesión por contraseña, logrando así que los usuarios de ésta puedan autenticarse de forma automática y segura. El motivo por el cual se quiere utilizar Chaffing and Winnowing es por su confidencialidad y rapidez sin la necesidad de cifrar la información [3].

⁵Un dato sensible es un dato que requiere una mayor protección que el resto de datos de carácter personal.

1.3.1. Limites y alcances

Los limites y alcances que tienen este trabajo terminal son los que se enlistan a continuación:

Límites

- Los certificados que se usan para este trabajo son auto firmados debido a que no se cuenta con una autoridad certificadora de confianza.
- Las pruebas se realizarán en un Servicio Web local y en fase de desarrollo, debido a que no contamos con un servidor o servicio de hosting.
- El Servicio Web que deseé implementar nuestro sistema debe de modificar su lógica de negocio de inicio de sesión para poder utilizar la API desarrollada en este trabajo.
- El sistema no cuenta con una recuperación de contraseña para los usuarios que se registren en la extensión.
- Es necesario que la extensión pueda conectarse con el servidor autenticador para el correcto funcionamiento del sistema.

Alcances

- Este sistema es ofrecido como alternativa para un inicio de sesión (mediante usuario y contraseña) dedicado a páginas web que tengan este método de autentificación.
- El sistema es ofrecido como una alternativa para el inicio de sesión para los servicios web que cuenten con autentificación mediante usuario y contraseña.
- La probabilidad de que se repita alguno de los certificados que se emiten es $1/2^{256}$.
- El sistema funciona únicamente sobre los servicios web que utilicen la API desarrollada.
- Gracias a la interoperatividad de los navegadores, la extensión podrá ser utilizada en los navegadores Opera a partir de la versión 63.0.3368.107 y Google Chrome de la versión 79.0.3928.

1.4. Metodología.

El proceso de desarrollo que seguiremos estará basado en la metodología de **"Prototipos Evolutivos"** con unas pequeñas adaptaciones que realizamos para que pueda funcionar mejor en el proceso de realización del sistema. Esta metodología ayuda a los desarrolladores a mejorar la comprensión del trabajo terminal a elaborar cuando los requerimientos no están completamente definidos o pueden ir cambiando a lo largo de la implementación . Así mismo, el paradigma nos brinda la posibilidad de utilizar fragmentos de programas existentes o aplicar herramientas que nos permitan generar rápidamente proyectos que funcione y puedan evolucionar [24].

Al utilizar este paradigma de desarrollo se busca que, mediante la implementación parcial del trabajo, vayan surgiendo requerimientos no vistos desde el planteamiento del problema, de esta manera, nos es posible ir experimentando con un prototipo parcialmente funcional e identificar posibles mejoras o fallas, con el fin de lograr el objetivo final.

En la Figura 1.1, se muestra un diagrama de las fases de esta metodología. Primeramente, tenemos la fase de 'Comunicación' en donde el equipo de desarrollo se reúne para definir los objetivos generales del trabajo terminal, se identifican los requerimientos conocidos y se detectan las áreas de mayor atención. Después, se llevan a cabo las fases 'Plan rápido' y 'Diseño rápido' en donde se hace un análisis del prototipo actual y se modela para dar paso al desarrollo del prototipo ('Construcción del prototipo'). Finalmente, para terminar una iteración, se realiza la fase 'Despliegue, entrega y retroalimentación' en donde se evalúa el funcionamiento realizando pruebas que nos brindan una retroalimentación para mejorar los requerimientos [24].

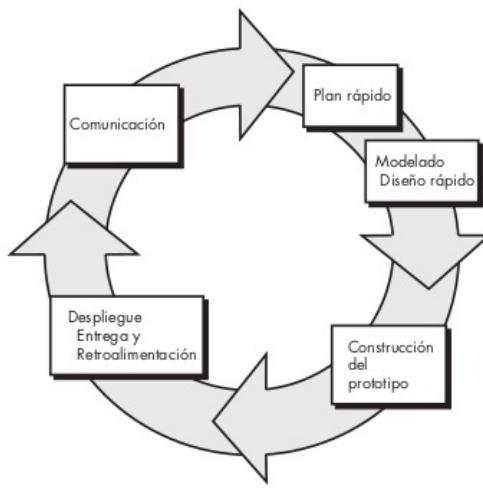


Figura 1.1: Diagrama de los fases de la metodología de prototipos evolutivos.

Para nuestro trabajo terminal y como primera fase vamos a realizar un prototipo con el fin de evaluarlo y considerar mejoras para la siguiente evolución del mismo. No se conoce el número total de prototipos que se deban realizar y para evaluarlos será necesario implementar cada uno de estos, es importante resaltar que nuestro sistema se encuentra conformado por componentes y que en algunos casos existe dependencia por lo que es indispensable considerar que para el correcto funcionamiento del prototipo se debe realizar un componente mucho antes que otro.

A continuación se indica cada una de las etapas de la metodología con lo que se realizará en cada una de estas:

1. Comunicación y plan rápido
 - Reunir requerimientos del prototipo
 - Componente 1
 - Investigar sobre el desarrollo de extensiones en Google Chrome.
 - Investigar sobre el método Chaffing and Winnowing.
 - Componente 2
 - Investigar sobre Certificados
 - Investigar sobre Autoridades Certificadoras.
 - Componente 3
 - Investigación sobre el servidor Apache.

- Analizar la arquitectura del servidor Apache
- Investigación sobre la versión de Apache conveniente a modificar.
- Describir el prototipo.

2. Modelado y diseño rápido

- Componente 1
 - Modelar un algoritmo que genere un patrón de chaffing.
 - Modelar un algoritmo de chaffing.
 - Crear un método que reciba un certificado, una cabecera HTTP y el patrón.
 - Diseñar una cabecera HTTP modificada que pueda contener la información de autenticación
- Componente 2
 - Diseñar una Base de datos para almacenar los datos del usuario y su certificado.
- Componente 3
 - Diseñar una API en el servicio web que permita realizar la etapa de winnowing.
 - Diseñar una arquitectura del sistema.

3. Construcción del prototipo

- Componente 1
 - Implementar el algoritmo de chaffing.
 - Implementar una extensión que intercepte peticiones HTTP.
 - implementar un login en donde el usuario pueda iniciar sesión.
- Componente 2
 - Implementar la Autoridad Certificadora.
 - Implementar un Servidor que reciba y envíe peticiones.
- Componente 3
 - Implementar un servidor Apache.
 - Implementar una API que reciba una petición y pueda identificarla.

4. Despliegue, entrega y Retroalimentación:

Después de la implementación del prototipo se analizará si puede haber mejoras y si es conveniente la realización de otro prototipo evolucionando este o desechándolo.

1.5. Organización del documento

Para dar inicio a este trabajo terminal, presentamos de manera breve la estructura de este reporte, con el objetivo de que el lector pueda tener un mejor entendimiento del trabajo.

1.5.1. Capítulo 2. Marco teórico.

Dado que este trabajo terminal relaciona temáticas muy enfocadas a la seguridad y aspectos web, es necesario conocer algunos conceptos e ideas fundamentales tanto para entenderlo como para conocer su funcionamiento, por lo tanto será necesario hablar del protocolo HTTP, métodos de autenticación, criptografía y acerca del método *Chaffing and Winnowing* que es la parte medular de todo este trabajo; en este marco teórico se explicará de manera breve y con un enfoque directo el uso que le daremos a estos conceptos en el trabajo terminal.

1.5.2. Capítulo 3. Análisis.

Dentro de este capítulo se analiza el estudio de factibilidad tanto técnico como operativo y económico con la finalidad de conocer los recursos necesarios para la elaboración de este trabajo terminal. Se mencionan de manera resumida las herramientas a utilizar y se explica de manera general la arquitectura del sistema y el diagrama de casos de uso general. Finalmente se muestra el análisis de los 3 componentes que tenemos en el sistema.

1.5.3. Capítulo 4. Diseño.

En el tercer capítulo, no adentraremos en el desarrollo de los prototipos, es decir, se encuentran los diagramas pertinentes para poder modelar nuestro trabajo terminal y proceder a la etapa de codificación. En este capítulo se desarrollan y explican los siguientes diagramas: 'Casos de uso', 'Flujo', 'Flujo de datos', 'Clases', 'Secuencia' y 'Actividades' y se muestra la interfaz de usuario propuesta junto con los requisitos de diseño.

1.5.4. Capítulo 5. Desarrollo.

En este capítulo, mostraremos lo que hemos desarrollado para este TT1 (Componente I). Se muestra que el prototipo cumpla los requerimientos que se le impusieron en la sección de análisis.

1.5.5. Capítulo 6. Avances y trabajo por hacer.

En el último capítulo de este reporte, hablaremos de los avances que hemos logrado a lo largo de la asignatura de trabajo terminal I y además, exponemos el trabajo esperado para la asignatura de trabajo terminal II.

Capítulo 2

Marco teórico

2.1. Protocolo HTTP

Desde 1990, el protocolo HTTP (Hypertext Transfer Protocol, Protocolo de transferencia de hipertexto) es el protocolo más utilizado en Internet. La versión 0.9 solo tenía la finalidad de transferir los datos a través de Internet (en particular páginas web escritas en HTML). La versión 1.0 del protocolo (la más utilizada) permite la transferencia de mensajes con encabezados que describen el contenido de los mensajes mediante la codificación MIME.

El propósito del protocolo HTTP es permitir la transferencia de archivos (principalmente, en formato HTML), entre un navegador (el cliente) y un servidor web (denominado, entre otros, httpd en equipos UNIX) localizado mediante una cadena de caracteres denominada dirección URL.

La comunicación entre el navegador y el servidor al que se quiere acceder se realiza de la siguiente manera en la figura 2.1:

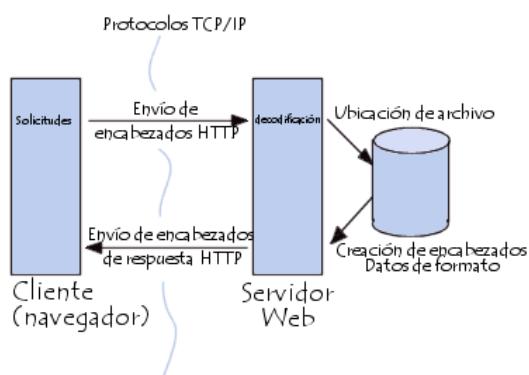


Figura 2.1: Comunicación Cliente - Servidor en el protocolo HTTP

El navegador realiza una solicitud HTTP y el servidor procesa la solicitud y después envía una respuesta HTTP. En realidad, la comunicación se realiza en más etapas si se considera el procesamiento de la solicitud en el servidor. Dado que sólo nos ocupamos del protocolo HTTP, no se explicará la parte del procesamiento en el servidor en esta sección del artículo [6].

2.1.1. Solicitud HTTP

Una solicitud HTTP es un conjunto de líneas que el navegador envía al servidor. Comprende:

- **Una línea de solicitud:** una línea que especifica el tipo de documento solicitado, el método que se aplicará y la versión del protocolo utilizada. La línea está formada por tres elementos que deben estar separados por un espacio: el método, la dirección URL y la versión del protocolo utilizada por el cliente (por lo general, HTTP/1.0)
- **Los campos del encabezado de solicitud:** un conjunto de líneas opcionales que permiten aportar información adicional sobre la solicitud y/o el cliente (navegador, sistema operativo, etc.). Cada una de estas líneas está formada por un nombre que describe el tipo de encabezado, seguido de dos puntos (:) y el valor del encabezado.
- **El cuerpo de la solicitud:** un conjunto de líneas opcionales que deben estar separadas de las líneas precedentes por una línea en blanco y, por ejemplo, permiten que se envíen datos por un comando POST durante la transmisión de datos al servidor utilizando un formulario.

2.1.2. Respuesta HTTP

Una respuesta HTTP es un conjunto de líneas que el servidor envía al navegador. Está constituida por:

- **Una línea de estado:** una línea que especifica la versión del protocolo utilizada y el estado de la solicitud en proceso mediante un texto explicativo y un código. La línea está compuesta por tres elementos que deben estar separados por un espacio: la versión del protocolo utilizada, el código de estado y el significado del código.
- **Los campos del encabezado de respuesta:** un conjunto de líneas opcionales que permiten aportar información adicional sobre la respuesta y/o el servidor. Cada una de estas líneas está compuesta por un

nombre que califica el tipo de encabezado, seguido por dos puntos (:) y por el valor del encabezado.

- **El cuerpo de la respuesta:** contiene el documento solicitado.

2.2. Certificado

Un certificado digital (X.509, v.3) es un fichero que contiene (entre otros datos) la clave pública (y privada) de una persona y está avalado (firmado electrónicamente) por una entidad de confianza o *Autoridad de Certificación*.

La Fabrica Nacional de Moneda y Timbre (FNMT) emite certificados digitales (los certificados del Documento Nacional de Identidad [DNI] están emitidos por la Dirección General de la Policía - Fabrica Nacional de Moneda y Timbre [DGP-FNMT]) [33].

El certificado digital contiene, entre otros datos, los siguientes:

- Filiación del propietario (Nombre, DNI, email).
- Protocolo de firma que se lleva a cabo (C,D).
- Autoridad de certificación que emite el certificado (FNMT).
- Fecha de emisión y de caducidad (2-3 años).
- Clave publica -privada- del propietario ($K -k-$).
- Firma electrónica de la autoridad de certificación.

Todo certificado debe estar firmado electrónicamente por una Autoridad de Certificación (AC) para ser valido. Este proceso de firma lo lleva a cabo la AC a la vez que genera el certificado (Figura 2.2), se lleva a cabo de la siguiente manera:

- Recopila los datos del usuario y las claves generadas por su navegador.
- Calcula un resumen (hash) de todos los datos recopilados. Firma electrónicamente el resumen anterior y lo une a los datos recopilados (Valor de la firma del certificado o Huella digital).

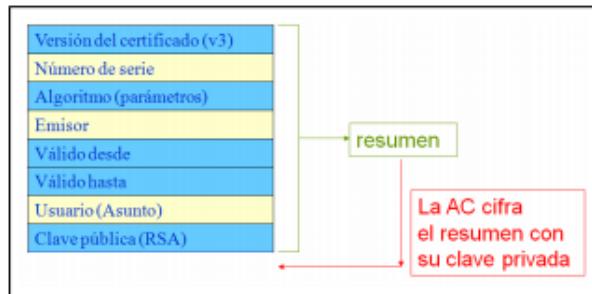


Figura 2.2: Firma de un certificado por una AC.

Antes de que un certificado personal (el único que puede tener claves privadas) sea instalado en el almacén de un navegador, éste verifica que está firmado por una AC (Figura 2.3) de las incluidas en su almacén de certificados (Autoridades) de la siguiente manera:

- Calcula el resumen (hash) de todos los datos recopilados del certificado, sin incluir la firma electrónica de la Autoridad Certificadora.
- Descifra la firma del certificado con la clave pública de la AC (debe estar en su almacén de Autoridades).
- Compara el resumen calculado con lo descifrado y decide sobre su autenticidad.

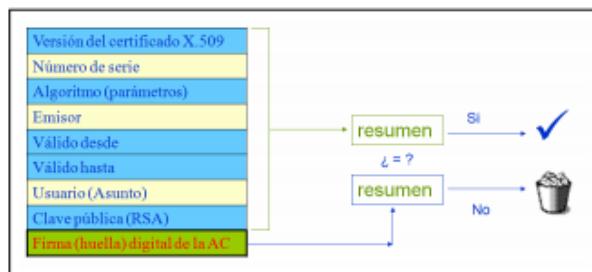


Figura 2.3: Verificación de un certificado firmado por una AC.

2.2.1. Autoridad Certificadora.

En una infraestructura de llave pública, se utilizan certificados digitales que son emitidos por una entidad de confianza, esta se encarga de validar y asegurar la identidad y validez de los certificados emitidos. Una Autoridad

Certificadora (CA por sus siglas en inglés *Certificate Authority*) es uno de los componentes de esta infraestructura, la CA es conocida por dos atributos, su nombre y su llave pública y lleva a cabo 3 funciones principales:

1. Emite certificados: crea los certificados de los subscriptores y los firma.
2. Mantiene la información del estado de los certificados y permite la validación del estado de los certificados emitidos.
3. Mantiene archivos de la información de los estados de los certificados, ya sea expirados o revocados, que este mismo emitió.

La Autoridad Certificadora es el tercero de confianza que nos garantiza la autenticidad de los certificados que son emitidos y recibidos.[46]

2.3. Métodos de autenticación en internet.

Con la evolución de la web, distintas páginas ofrecen ciertos servicios a los usuarios y con la finalidad de dar una experiencia óptima y segura, se requiere que una persona o usuario se identifique para el uso de estos servicios, es aquí donde entra en juego el papel de los métodos de autenticación. Para poder asegurar la confidencialidad de la información manejada en todos estos servicios, es necesario restringir el acceso de este, para esto se utiliza la identificación y la autenticación; la identificación es un procedimiento donde el sujeto es reconocido por algún ID, esto es equivalente al saber una parte de información en específico, mientras que la autenticación es el proceso de validación de si el sujeto es la persona quien dice ser al tratar de identificarse [17].

Para probar una identidad, el sujeto presenta algo llamado "factor de autenticación", principalmente existen 4:

- El sujeto tiene algo (Token, documento, un material específico, etc.).
- El sujeto conoce algo (contraseña, login, respuesta a una pregunta, etc.).
- El sujeto tiene una característica biológica (huella dactilar, ADN, etc.).
- El sujeto se encuentra en un lugar en específico (dirección IP, información de un lugar en específico, etc.).

Hoy en día, la autenticación por contraseña es el método más utilizado, más que otra cosa por su ventaja principal: su simplicidad de utilización, sin

embargo, así como tiene una gran ventaja, la autenticación por contraseña también tiene muchos problemas y desventajas de seguridad.

A continuación en el cuadro 2.1, mostraremos algunas tablas comparativas que servirán para tener una mejor perspectiva de las ventajas, desventajas, vulnerabilidades de los diferentes métodos de autenticación, entre otras cosas:

	Recordar	Otros dispositivos	Acciones	Facilidad	Tiempo	Errores	Recuperación
Contrasenñas	1	3	2	3	3	2	3
Otros recursos	2	3	3	3	3	3	2
Contrasenñas gráficas	1	1	2	3	3	2	3
Contrasenñas dinámicas	1	3	2	2	3	2	2
Tokens	3	1	1	2	2	3	1
Multivariación	1	1	1	3	2	2	1
Cryptografía	3	1	1	1	1	2	1
Biométricos	3	3	2	3	2	2	1

Cuadro 2.1: Tabla comparativa de la aplicación en los distintos métodos de autentificación

La tabla anterior concentra las siguientes características:

- Recordar: Hace referencia a que tan complicado es que un usuario se acuerde de los datos necesarios para la autentificación.
- Otros dispositivos: El usuario usa una entidad externa para facilitar su autentificación.
- Acciones: Hace referencia a que tantas acciones adicionales se deben de realizar para autenticarse.
- Facilidad: Simplicidad de tecnología.
- Tiempo: Cantidad de recursos temporales que consume el método de autentificación.
- Errores: Posibles errores durante la autentificación.
- Recuperación: Denota la dificultad de recuperar la clave de acceso en caso de pérdida.

En el cuadro 2.2 se muestra una tabla comparativa del nivel de seguridad en los distintos métodos de autentificación, donde 1 - baja seguridad, 2 – media seguridad y 3 – alta seguridad.

	Ataque por fuerza bruta	Observación	Hackeo indirecto	Phishing
Contraseñas	1	1	1	1
Otros recursos	2	2	3	3
Contraseñas gráficas	1	1	2	2
Contraseñas dinámicas	2	3	2	2
Tokens	3	3	3	3
Multivariación	1	1	3	3
Criptografía	3	3	3	3
Biométricos	3	3	1	1

Cuadro 2.2: Tabla comparativa de la seguridad en los distintos métodos de autentificación

La tabla se enfoca principalmente en los siguientes problemas de seguridad:

- Ataque por fuerza bruta: Se descifra el método de autentificación con una gran cantidad de intentos, usualmente generados por un programa.
- Observación: Cuando se intenta ver directamente los datos necesarios para la autentificación desde una distancia cercana hasta incluso usando binoculares, cámaras o algún otro dispositivo.
- Hackeo indirecto: El usuario confía sus datos del método de autentificación a terceros quienes pueden ser atacados.
- Phishing: Hace referencia a programas que se hacen pasar por entidades confiables para interceptar los datos que desean.

Seguridad en internet En la actualidad, el incremento constante de internet ha impactado directamente en la seguridad de la información que se maneja cotidianamente y por la mayoría de usuarios. Existen infinidad de sitios donde es aplicada la seguridad, ya que sin ésta, se verían afectados todos los usuarios en sus cuentas, pudiendo verse afectados desde un posible Identity theft (Robo de identidad), hasta la pérdida de dinero real dado que la base de algunas de estas páginas son E-Commerce, estas páginas implican el manejo de tarjetas de crédito, paypal, etc.

Uno de los puntos más críticos de la seguridad en Internet son las herramientas que interactúan de forma directa con los usuarios. Es común escuchar sobre fallas en los sistemas de protección de los servidores más frecuentemente utilizados, por ejemplo Apache, NGINX, IIS, etc. O en los lenguajes de

programación en que son escritas las aplicaciones [26]. Sin embargo, la vulnerabilidad más grande dentro de un sistema, son los ataques directos a los usuarios finales durante la autentificación.

Una de las técnicas de autentificación que actualmente se usa es "recordar la sesión" usando las "cookies", en la siguiente sección, nos adentraremos en definir qué son las cookies y exponer sus vulnerabilidades.

2.4. Cookies.

Durante la navegación por internet, la información sobre la computadora puede ser colectada y almacenada. Ésta puede ser de carácter general sobre el equipo y puede ser también información más específica sobre los hábitos de navegación del usuario, toda esta información guardada se le conoce como Cookies.

A continuación se muestran los diferentes tipos de cookies que existen para los navegadores.

- **Cookies propias:** Las cookies se gestionan desde el terminal o dominio de un mismo editor.
- **Cookies de terceros:** Las cookies no son enviadas por el propio editor, sino por otra entidad.
- **Cookies de sesión:** Los datos recabados sólo se recogen mientras el usuario está navegando por la página web.
- **Cookies persistentes:** Los datos continúan almacenados en el terminal y se puede acceder a ellos durante un periodo de tiempo determinados.
- **Cookies técnicas:** Permiten controlar el tráfico y la comunicación de datos.
- **Cookies de personalización:** Dejan a los usuarios acceder según algunas características propias que se recogen (navegador, idioma, etc.).
- **Cookies de análisis:** Recogen datos sobre el comportamiento de los usuarios y permiten elaborar un perfil de usuario.
- **Cookies publicitarias:** Recogen datos sobre la gestión de los espacios publicitarios.

Las cookies persistentes son aquellas que se almacenan en el equipo para que las preferencias personales puedan ser retenidas, ayudan a los sitios web a recordar tu información y ajustes cuando los visitas más adelante. Esto conlleva un acceso más rápido y sencillo ya que, por ejemplo, no se tiene que iniciar sesión de nuevo. Además de la autentificación, otras páginas web tienen más funciones para las cookies permanentes, como: selección de idioma, selección de tema, preferencias de menú, marca-páginas internos de la web, o favoritos. Muchos navegadores pueden ajustar el periodo de tiempo en que las cookies persistentes deben ser almacenadas. [34]

Gracias a las cookies persistentes, las direcciones de correo electrónico aparecen por default cuando se abre el correo electrónico, o en páginas de inicio personalizadas cuando se visita en línea un comercio. Si un atacante obtiene acceso puede recopilar información personal del usuario través de estos archivos y poder robar toda información del usuario. Es fácil acceder a estas cookies y obtener fácilmente la información del usuario, por lo que es necesario que el usuario nunca deje vulnerable esta información o en su debido caso borrar cookies al término de cada sesión. Existen diferentes funcionalidades para las cookies, una de las más importantes es la funcionalidad de seguridad, ya que contiene información importante de los usuarios [34]. A continuación se muestran las diferentes funcionalidades de las cookies.

- **Preferencias:** Sirven para que la página se visualice atendiendo a los gustos del usuario, como por ejemplo idioma, región o tamaño de textos.
- **Seguridad:** Se encargan de autenticar a los usuarios y evitar el uso fraudulento de las credenciales por parte de terceros.
- **Procesos:** Son utilizadas para el correcto funcionamiento de la página en el navegador.
- **Publicitarias/Estadísticas:** Se usan para que la publicidad que se muestre sea personalizada.
- **Estados de la sesión:** Obtiene información del comportamiento del usuario en una página web, como por ejemplo el tiempo que pasa en una página, los "clicks" que realiza o la publicidad que le aparece.

Una vez entendido que es la autentificación y el uso que se le puede dar a las cookies, procedemos a exponer como la criptografía es de gran ayuda en la seguridad en internet y como la usaremos para este nuevo método de autentificación.

2.5. Introducción a la Criptografía.

2.5.1. Criptología.

Para comprender que es la criptografía, es necesario que comprendamos qué es la "*Criptología*", palabra que proviene del griego «*kryptós*» (oculto) y «*logos*» (estudio). Según la Real Academia Española significa 'estudio de los sistemas, claves y lenguajes ocultos o secretos', sin embargo, brindándole un contexto a esta definición decimos que es el arte y ciencia que se encarga de diseñar sistemas para ocultar mensajes, y buscar la manera de romper dichos sistemas [4].

La criptología contiene dos ramas, las cuales son: el criptoanálisis y la criptografía. Ésta última es de vital importancia en este trabajo terminal, por lo que a continuación se explicará qué es, su historia, sus objetivos y sus usos que tiene esta rama en la actualidad.

2.5.2. Criptografía.

"*Criptografía*", palabra proveniente del griego «*kryptós*» (oculto) y «*graphos*» (escribir), es definida por la Real Academia Española como 'el arte de escribir con clave secreta o de un modo enigmático'. Nuevamente, la definición de la RAE nos da un panorama general de lo que trata esta rama de la criptología, sin embargo, en la actualidad tenemos definiciones más extensas y precisas que nos ayudan a entender las funciones de este arte.

A continuación, se presentan dos definiciones de la criptografía, cabe mencionar que estas definiciones están orientadas al uso de la criptografía en la informática y las telecomunicaciones actualmente.

Jorge Ramírez Aguirre nos brinda la siguiente definición en su libro 'Seguridad informática y criptografía' [2].

"Rama inicial de las matemáticas y en la actualidad también de la informática y la telemática, que hace uso de métodos y técnicas con el objeto principal de cifrar, y por tanto proteger, un mensaje o archivo por medio de un algoritmo, usando una o más claves."

Finalmente, Menezes, Van Oorschot y Vanstone, no brindan en su libro una definición formal de lo que es la criptografía (traducción) [21].

"Estudio de técnicas matemáticas relacionadas con los aspectos de la seguridad de la información tales como la confidencialidad, la integridad de datos, la autenticación de entidad y del origen de datos. La criptografía no

comprende sólo a los medios para proveer seguridad de información, sino a un conjunto de técnicas.”

A elección del lector elegir aquella definición que le convenza más, nosotros una vez finalizado la exposición de estas definiciones de criptografía, continuaremos con una breve remembranza de la historia de esta disciplina.

En egipto hace 4000 años, tuvo su primera aparición la criptografía, cuando un maestro egipcio escribió la historia de su señor utilizando jeroglíficos poco comunes tratando de imprimir cierta jerarquía. Este primer acercamiento a la criptografía, utilizaba las técnicas de sustitución y transposición de símbolos de una manera similar a la base del concepto general de cifrado. Posteriormente, las antiguas civilizaciones occidentales comienzan a adoptar estas técnicas para mantener determinada información oculta, principalmente con propósitos militares, diplomáticos y religiosos. Mientras la criptografía crecía alrededor del mundo, el criptoanálisis también lo hacía, con el objetivo de hallar el mensaje original a partir de un mensaje cifrado, sin conocer el método utilizado [7] [19].

En la historia conocida después de lo antes mencionado, existe un punto crucial en el desarrollo de la criptografía tal y como la conocemos hoy en día hasta la llegada de las computadoras. Este punto fue la Segunda Guerra Mundial, en donde se construyeron máquinas de cifrado mecánicas y electromecánicas que aceleraban el proceso de cifrado y descifrado. Los alemanes desarrollaron la famosa máquina 'Enigma', que precisamente mediante unos rotores automatizaba el proceso de cifrado y descifrado de mensaje, brindándole al ejército una ventaja considerable en la inversión de tiempo en la comunicación.

2.5.2.1. Criptografía moderna.

En la actualidad, el término '*criptografía moderna*' hace alusión al desarrollo de esta ciencia en las áreas de la teoría de la información y las comunicaciones. Claude Shannon, considerado por muchos como el padre de la criptografía matemática, en su libro "Sistemas secretos" estableció las bases para las implementaciones de los algoritmos actuales a mediados de los años 50s [19].

En los años 70s, el público general tuvo acceso al trabajo de Claude, además surgió la llegada de las computadoras y la publicación el primer borrador del algoritmo de criptografía simétrica DES, el cual fue el primer algoritmo público, basado en técnicas matemáticas y criptográficas modernas. Todo ello

representó los cimientos para un rápido crecimiento en la criptografía, hasta eventualmente llegar a otro hecho sumamente relevante que determinó gran parte de las transacciones que realizamos hoy en día en internet. Dicho hecho fue el artículo de las nuevas direcciones de criptografía hecho por Whitfield Diffie y Martin Hellman, el cual trataba de un nuevo método para distribuir llaves, que eventualmente se llamaría criptografía asimétrica.

2.5.3. Objetivos de la criptografía.

Algunos de los objetivos que se busca con la implementación de la criptografía para la seguridad de la información son los siguientes:

- **Confidencialidad:** este objetivo, también conocido como privacidad de la información, implica mantener en secreto una determinada información, por tanto, sólo aquellas personas que estén autorizadas tendrán acceso a ella.
- **Autentificación:** este objetivo, implica hablar de la corroboración de la identidad de una entidad, por tanto, asegura que la entidad de donde la información es originada pueda ser identificada.
- **Integridad:** este objetivo asegura que determinada información no haya sido alterada por personas no autorizadas o por cualquier otro medio no conocido.
- **No repudio:** este objetivo previene que una entidad niegue un envío de información de un acuerdo preestablecido.

2.5.4. Usos de la criptografía.

Los usos que tiene la criptografía son muy variadas, dependiendo del ámbito donde se está aplicando. Las siguientes usos, son sólo algunas de los tantos que hay y provienen de distintos objetivos que tiene la criptografía aplicada a la seguridad de la información [19].

- **Autorización:** permiso concreto, de una parte o entidad, para el acceso o la realización de una tarea específica.
- **Validación:** proveer una autorización para el uso o la manipulación de información o recursos.
- **Control de acceso:** restricción de acceso a la información o recurso.

- **Certificación:** respaldo de información por una entidad externa de confianza.
- **Confirmación:** acuse de recibo a servicios que han sido dados.
- **Anonimato:** ocultamiento de la identidad de una entidad.

2.5.5. Criptografía simétrica y asimétrica.

Anteriormente en la historia de la criptografía, se hizo una rápida mención del nacimiento de estos dos métodos de cifrado, en esta sección se explicará más profundamente sus funciones con el fin de que el lector conozca un poco más y entienda porque hemos decidido utilizar determinado método para cumplir con los objetivos de este trabajo.

2.5.5.1. Criptografía simétrica.

En la criptografía simétrica, tanto el emisor como el receptor comparten una única llave secreta para cifrar y descifrar la información que se deseé transmitir. Esto implica que ambas partes de la comunicación deben tener un acuerdo antes de que se realice la comunicación. La seguridad de este tipo de algoritmos radica en mantener segura la llave secreta, por tanto, si ésta es revelada, cualquiera con acceso a ella puede descifrar el mensaje. Por estas razones, este tipo de criptografía puede ser visto como 'criptografía de llave privada'. Algunos de los algoritmos más famosos de criptografía simétrica son: DES (Data Encryption Standard), TripleDES, AES (Advanced Encryption Standard) y IDEA (International Data Encryption Algorithm) [19]. En el esquema de la Figura 2.4, se muestran los pasos que sigue un protocolo de criptografía simétrica. Definamos 'A' como una entidad que pretende enviar un mensaje a otra entidad llamada 'B'. Luego entonces, ambas partes acordarán una 'Llave Secreta' con la que 'A' cifrará el mensaje utilizando un algoritmo establecido, mandando el resultado (Texto Cifrado) a 'B' que descifrará el mensaje con la misma llave y algoritmo que 'A'.

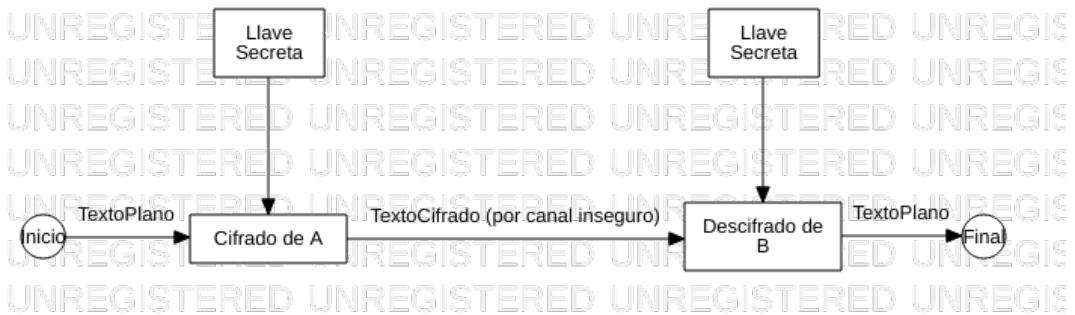


Figura 2.4: Esquema del protocolo de criptografía simétrica.

2.5.5.2. Criptografía asimétrica.

La criptografía de clave pública fue inventada en 1976 por los matemáticos Whitfield Diffie y Martin Hellman y es la base de la criptografía moderna. En los algoritmos de criptografía asimétrica, el receptor posee una llave pública y una llave privada para poder descifrar los mensajes. Las claves privadas deben ser conocidas únicamente por su propietario, mientras que la correspondiente clave pública puede ser dada a conocer abiertamente. Si un usuario quiere enviar a otro un mensaje de forma que sólo el receptor pueda entenderlo, lo codificará con la clave pública del receptor y éste utilizará su clave privada, que solo él tiene, para poder leerlo. Dicho esto, podemos llamar llave de cifrado a la llave publica y llave de descifrado a la llave privada, siendo ambas totalmente diferentes una de la otra. Algunos de los algoritmos más famosos de criptografía asimétrica son: RSA y ElGamal [19].

La criptografía asimétrica está basada en la utilización de números primos muy grandes. Si multiplicamos entre sí dos números primos muy grandes, el resultado obtenido no puede descomponerse eficazmente, es decir, utilizando los métodos aritméticos más avanzados en las computadoras más avanzadas sería necesario utilizar durante miles de millones de años tantas computadoras como átomos existen en el universo. El proceso será más seguro cuanto mayor sea el tamaño de los números primos utilizados.

En el esquema de la Figura 2.5, se muestran los pasos que sigue un protocolo de criptografía asimétrica. Definamos 'A' como una entidad que desea enviar información a otra llamada 'B'. Luego entonces, 'B' enviará a 'A' su llave pública para que 'A' cifice la información utilizándola. Cuando la información (Texto Cifrado) haya viajado a través del canal inseguro para que 'B' la reciba, 'B' descifrará el Texto Cifrado con su llave privada.

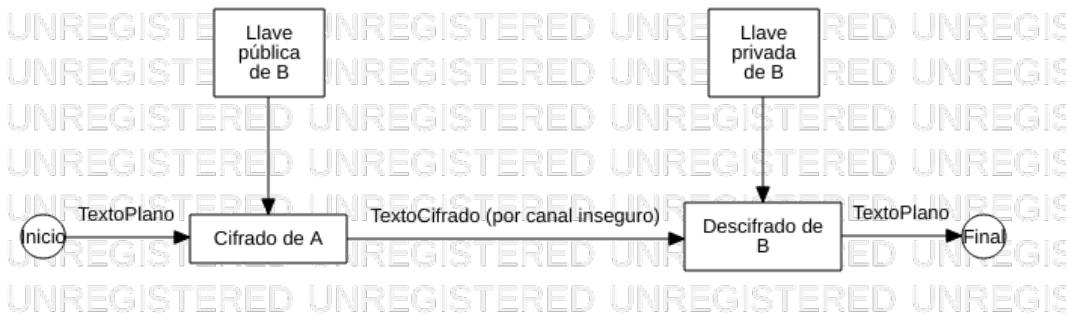


Figura 2.5: Esquema del protocolo de criptografía asimétrica.

Algoritmo RSA.

RSA es el algoritmo de cifrado asimétrico más popular en la actualidad. Creado por Ron Rivest, Adi Shamir y Leonard Adleman y publicado en el año 1977. El algoritmo es considerado seguro, en tanto sean utilizadas llaves de longitud suficientemente seguras (se siguen utilizando llaves de 1024 bits, pero ya se recomienda al menos una longitud de 2048). El algoritmo sirve tanto para cifrar y descifrar, así como también para la generación de firmas digitales. Es, en la actualidad, ampliamente utilizado en protocolos de comercio electrónico. La seguridad *RSA* está basada en la dificultad de realizar el *factoreo* de números grandes. La llave privada y la pública son generadas o calculadas en función de un par de números primos, del orden de los 200 dígitos o más grandes aún [19].

Al describir ya concretamente al algoritmo, se establece que para la generación del par de llaves (llave publica y privada) se deberán seleccionar dos números primos grandes aleatorios, p y q , y se calculará n como su producto:

$$n = pq$$

La llave de cifrado, e , será elegida también de manera aleatoria, tal que e y $(p-1)(q-1)$ sean primos relativos.

La llave de descifrado d será obtenida despejando la ecuación:

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

En otras palabras, o mejor dicho, en otros símbolos:

$$d = e^{-1} \operatorname{mod}((p-1)(q-1))$$

Los números e y n componen la llave privada; el número d corresponde a la llave privada; p y q serán descartados pero no revelados.

A la hora de cifrar un mensaje m , éste deberá ser dividido en bloques más pequeños que n y cada parte del texto-cifrado, c , será obtenida mediante:

$$c_i = m_i^e \bmod n$$

Para el descifrado, cada parte o bloque del texto-cifrado se tomará para calcular:

$$m_i = c_i^d \bmod n$$

El patrón con el que se genera el código *Chaffing* no puede viajar a través de la red visible para cualquier entidad que quiera obtener dicha petición, por lo que, debemos cifrarlo con algún algoritmo confiable.

El algoritmo RSA nos proporciona esa seguridad, debido a que es un cifrado asimétrico con el cual podemos cifrar el patrón con la llave publica del servicio. Una vez que la petición cifrada llegue al servicio, este será capaz de descifrar dicho mensaje con su llave privada, para así poder ver el mensaje original.

2.5.6. Criptografía a nivel de aplicación.

La criptografía a nivel de aplicación se entiende como aplicaciones o programas informáticos que hacen uso de diferentes técnicas criptográficas. Al hablar de criptografía en el nivel de aplicación, también podría considerarse a los protocolos criptográficos de alto nivel, los cuales determinan como han de comunicarse ambas partes, que algoritmos usar, define formatos, etc [19].

2.5.6.1. SSL/TLS.

Secure Sockets Layer (SSL) provee servicios de seguridad entre la capa TCP y las aplicaciones que hacen uso de esa capa. Actualmente, la sucesora de SSL es TLS (Transport Layer Service), sin embargo, lo acuñado que está el término SSL hace que se use indistintamente para referirse a TLS, aunado a ello, las diferencias entre la última versión de SSL (SSL3.0) y la primera versión de TLS (TLSv1) son menores, por lo que en el desarrollo de este reporte, utilizaremos el término SSL/TLS.

SSL/TLS provee entonces confidencialidad, lográndola con criptografía asimétrica y controlando la integridad de los datos utilizando un MAC (Message Authentication Code).

El proceso de comunicación del protocolo establece, como primer paso, la negociación de ambas partes de los algoritmos a utilizar. Luego, procede al intercambio de llaves públicas y a la autenticación basada en certificados digitales para, finalmente, cifrar de manera simétrica los datos o información a transferir [19].

En nuestro trabajo, usaremos SSL/TLS para brindar confidencialidad al canal de comunicación entre la extensión y el servidor autenticador. Esto se explicará más a detalle en el análisis del Componente II.

2.5.6.2. OpenSSL

OpenSSL es un proyecto de código abierto que implementa funciones criptográficas sin limitaciones dentro de una librería y que provee diversas herramientas útiles. OpenSSL actualmente implementa SSL2.0, SSL3.0 y TLSv [19].

Dentro de las herramientas que tiene se encuentran:

- Crear y manejar llaves privadas, públicas y parámetros.
- Realizar operaciones criptográficas de llave pública.
- Calcular hash de algún mensaje.
- Cifrar y descifrar con algoritmos simétricos.
- Crear certificados X.509 (CSRs y CRLs).

Esta última herramienta, es la que usaremos para este trabajo terminal. Se creará un certificado de cada usuario a partir de sus datos de inicio de sesión para autenticarlo en los servicios web en donde quiera acceder. Esto se explicará más a detalle en el análisis del Componente II.

Ahora que se ha expuesto el uso de la criptografía en este trabajo, procedemos a platicar acerca del método *Chaffing and Winnowing* y cómo se implementará en este trabajo.

2.6. Chaffing and Winnowing.

2.6.1. Historia

Chaffing and Winnowing es una técnica que logra confidencialidad sin usar ningún proceso de cifrado para el envío de datos sobre un canal inseguro. El nombre **Chaffing and Winnowing** el nombre proviene de la agricultura: Déspués de que el grano ha sido cosechado y trillado es mezclado con paja fibrosa no comestible. La paja y el grano son separados por el movimiento de las hojas y la paja es descartada. Ésta técnica fue creada por Ron Rivest y fue publicada en un articulo en línea el 18 de Marzo de 1998 [25]. Aunque parece ser similar a un cifrado tradicional o esteganografía, chaffing and winnowing no puede ser clasificado como uno de ellos.

Ésta técnica permite el envío de datos evitando la responsabilidad del cifrado de su contenido. Cuando se usa chaffing and winnowing, el emisor transmite el mensaje sin cifrar (texto plano). Aunque el emisor y el receptor comparten una llave, ellos la usan sólo para autenticar. Sin embargo, una tercera parte puede hacer su comunicación confidencial durante el envío simultáneo de mensajes especialmente mensajes diseñados a través del mismo canal.

2.6.2. ¿Qué es Chaffing and Winnowing?

Chaffing and Winnowing es un nuevo esquema establecido por Rivest en 1998. Este esquema ofrece confidencialidad para el contenido de un mensaje sin involucrarse con cifrado ni esteganografía [19].

El proceso **Chaffing** no hace uso de un cifrado por lo que no tiene una "clave de cifrado". Este proceso consiste en agregar paquetes inválidos (Información innecesaria) al mensaje a enviar, haciendo que el mensaje viaje seguro a la vista de todos los posibles "atacantes".

El proceso de **Winnowing** no emplea algún tipo de cifrado, por lo que al igual que el proceso chaff no tiene una "clave de descifrado". Intentando regular la confidencialidad que provee un cifrado damos paso a la esteganografía y el proceso de winnowing [25].

Existen dos partes en el envío de mensajes con winnowing: Autentificación (Agregando MACs) y agregando paquetes chaff. Nosotros nos enfocaremos mas al uso de paquetes chaff para el envío seguro de información, ya que, el receptor es quien remueve los paquetes chaff para obtener el mensaje original.

Los siguientes esquemas explican como es que se lleva a cabo el proceso de **Chaffing and Winnowing** en diferentes escenarios.

Escenario 1: Alice se está comunicando con Bob en un solo camino de comunicación sobre un canal inseguro y Charles agrega los paquetes de Chaff.

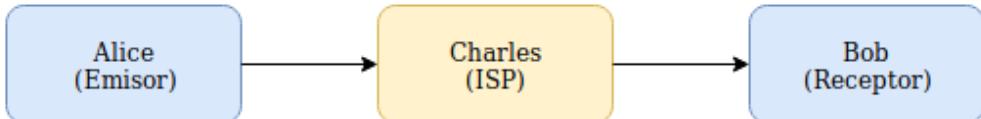


Figura 2.6: Charles agrega los paquetes inválidos.

En el escenario anterior Alice y Bob se están comunicando mutuamente por un canal de comunicación no seguro, en donde son enviados paquetes no cifrados. Alice y Bob comparten la llave de autentificación la cual será usada para el proceso de autentificación. Cuando Alice envía un mensaje a Bob, su mensaje es autenticado de su lado y es enviado a Charles antes de ser enviado a Bob. Charles agrega los paquetes chaff a la secuencia transmitida por Alice, al agregar los paquetes chaff, Charles provee confidencialidad para la comunicación entre Alice y Bob. Pero donde Charles no conoce la llave secreta compartida entre Alice y Bob. Por lo que el proceso de chaffing no necesita ningún conocimiento de la llave secreta de autentificación compartida.

Escenario 2: Alice se comunica con Bob en un camino de comunicación inseguro y en el cual Charles no agrega los paquetes chaff si no que multiplexa los flujos de las dos partes (David y Alice). Este escenario es diferente al anterior, ya que se multiplexa el flujo de datos de Alice y Bob con el flujo de datos de David y Jane, y cuando el paquete llega a Bob el flujo de paquetes de David hacia Jane es el chaff de Bob y es descartado y vice versa para Jane.

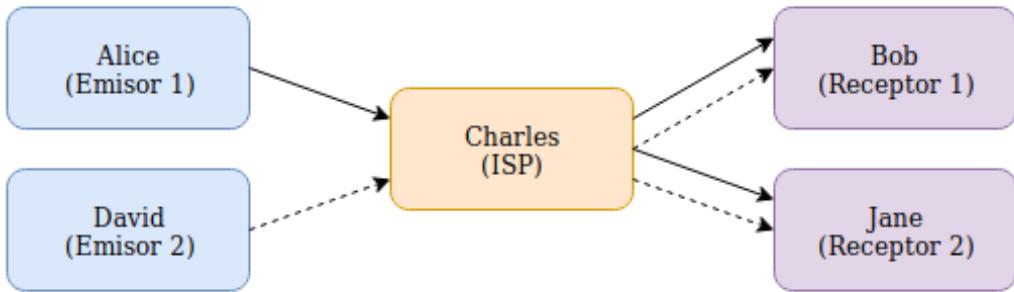


Figura 2.7: Charles no agrega los paquetes pero multiplexa los flujos.

Escenario 3: Alice se comunica con Bob en un canal de comunicación inseguro y Alice no agrega los paquetes chaff. En este escenario, Alice desarrolla la autentificación de sus mensajes, por lo que Alice aplica chaffing para autenticar los mensajes y producir una secuencia de paquetes que serán transmitidos a Bob por la vía de Charles. Bob lleva a cabo el proceso de winnowing para recuperar el mensaje original.

2.6.3. Objetivo de Chaffing and Winnowing

El objetivo de seguridad del esquema de chaffing-and-winnowing es proporcionar privacidad en un entorno simétrico. Desde un punto de vista de seguridad, este esquema debe tratarse simplemente como un esquema de cifrado simétrico. Hay algunos procesos de "cifrado" que toman un mensaje y crean un "texto cifrado", y algún proceso de "descifrado" toma el texto cifrado y recupera el mensaje, ambos operando bajo una clave secreta en común. (Para el esquema Chaffing and Winnowing es la clave para el MAC). Estos procesos no se implementan de manera "habitual", pero, de manera abstracta, deben existir, de lo contrario no se logra la privacidad [3] [11].

"No es una propiedad de seguridad novedosa, sino un conjunto novedoso de restricciones en los procesos dirigidos a lograr una propiedad de seguridad estándar"

"Encontrar-luego-adivinar". Extensión más directa al caso simétrico de la noción de indistinguibilidad.

Haciendo uso de **Chaffing and Winnowing** se asegura que los adversarios no obtengan información del mensaje transmitido a lo largo de un canal de comunicación inseguro entre dos partes.

Rivest propone un esquema, el cual cuenta con tres partes principales [25].

1. **Autentificación** Es el proceso de descomponer el mensaje original en un paquete más pequeño y complementar cada paquete con un código de autentificación de mensaje (MAC).
2. **Chaffing** Es el proceso de agregar paquetes inválidos (Chaff packets).
3. **Winnowing** Es el proceso de remover paquetes Chaff para obtener el mensaje original en texto plano.

2.6.4. ¿Cómo funciona?

El esquema de Chaffing and Winnowing deja que cada paquete conste de:

- Un número de serie
- Contenido del paquete
- Código de autentificación del mensaje

Cuando son enviados los paquetes, el mensaje con el texto plano se descompone en pequeños paquetes los cuales contienen datos y el tamaño del paquete original. Entonces, el emisor (Alice) usa el algoritmo de **código de autentificación de mensaje** (MAC) para generar el valor MAC para ser agregado al paquete y el cual se basa en el número de serie, contenido del paquete y la llave autentificación. La Figura 2.8, muestra la salida del paquete después del proceso de autentificación.

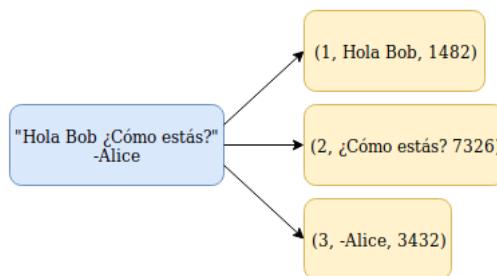


Figura 2.8: Secuencia de Chaffing después del proceso de autentificación.

Esta secuencia de paquetes es enviada a Charles (ISP) para llevar a cabo el proceso de Chaffing. Charles agrega paquetes chaff a la secuencia de paquetes antes de ser enviados por medio del canal de comunicación y ser recibidos por Bob.

Existen dos maneras donde Charles puede enviar la secuencia de chaff hacia Bob. La primera es enviando aleatoriamente mezclados los paquetes chaff para formar una secuencia y la otra manera es enviarlos de manera ordenada por el número de serie seguido del contenido del mensaje. En la Figura 2.9, se muestra cómo es el proceso de chaff en esta secuencia.

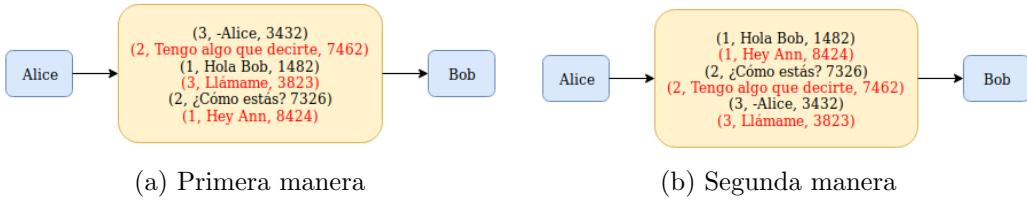


Figura 2.9: Las dos maneras para el proceso de chaff pueden ser utilizadas. Los paquetes chaff son los mensajes de color rojo.

Una vez que la secuencia de chaff llega a Bob, el último proceso es Winnowing. Bob determina la secuencia del mensaje que es válida del paquete chaff usando una función hash para el contenido de cada paquete y la llave de autentificación para re-calcular el MAC y compararlo contra el MAC del paquete recibido, si la comparación falla, el paquete chaff es descartado. Si la comparación es valida, entonces el paquete es parte del mensaje original. En la Figura 2.10, se muestra el proceso completo de Chaffing and Winnowing [25].

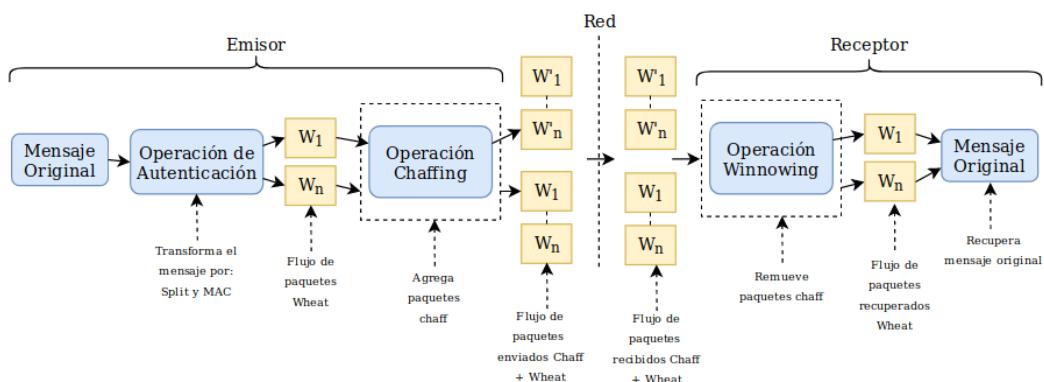


Figura 2.10: Visión general del proceso Chaffing and Winnowing.

2.6.5. Propiedades de Chaffing and Winnowing

- La técnica de Chaffing y Winnowing no depende de la fuerza del esquema de cifrado para proporcionar confidencialidad debido al hecho de que es muy difícil distinguir la información útil de los paquetes chaff sin la clave secreta. Por lo tanto, la dificultad de distinguir la información útil del chaff proporciona confidencialidad al esquema.
- La operación de Chaffing puede ser realizada por un tercero, ya que la clave secreta compartida no es necesaria en el proceso del mismo.
- Los paquetes de Chaff no tienen que contener datos aleatorios, ya que uno podría usar un mensaje válido con una clave secreta diferente para hacer el paquete de Chaff. Cuando el receptor recibe esos paquetes de Chaff, se verán como paquetes de Chaff, ya que la clave que se usa para volver a calcular el Chaff es diferente de la que los hace.

2.6.6. All-or-Nothing and the Package Transform (AONT)

All-or-Nothing and the Package Transform es una variación dentro de la técnica Chaffing and Winnowing, donde se mejora la eficiencia de su esquema original. AONT es la transformación de pre-procesamiento que permite a las partes enviar más datos (en términos de bit) por paquete en lugar de solo uno. Este pre-procesamiento es una transformación sin cifrado que toma el mensaje de texto sin formato y produce un mensaje empaquetado que luego se procesa de la manera normal de Chaffing and Winnowing [18]. Las definiciones de la transformación AONT son las siguientes:

1. El algoritmo de transformación es **reversible**: Dado el bloque de mensaje transformado, el receptor puede obtener el mensaje de texto sin formato original.
2. El algoritmo de transformación y su inverso son **computables** de manera eficiente: Lo que significa que es computacionalmente factible recrear el texto original dada la llave privada y recibir todos los paquetes con éxito.
3. La transformación no es **computacionalmente factible**: Esto significa que si se ha recibido parte del paquete de la transmisión, cualquiera que esté intentando leer el mensaje no puede hacerlo ya que la transformación **AONT** requiere que se reciba todo el mensaje, de lo contrario no entrega nada.
4. La transformación es una **técnica sin cifrado**: La técnica de pre-procesamiento no tiene llaves y no hay una llave secreta compartida

involucrada en la operación. Cualquier persona que haya recibido todos los mensajes transformados del paquete puede recuperar el mensaje de texto original.

¿Cómo funciona AONT?

Supongamos que el mensaje de entrada es el siguiente: m_1, m_2, \dots, m_n . Seleccionamos una llave aleatoria K' el cual se usará para la función del paquete de transformación. Se calcula la secuencia transformada m'_1, m'_2, \dots, m'_s para $s' = s + 1$ como se muestra a continuación:

Tenemos:

$$m_i \otimes E(K', i) \text{ for } i = 1, 2, 3, \dots, s$$

También:

$$m'_{s'} = K' \otimes h_1 \otimes h_2 \otimes \dots \otimes h_s$$

Donde:

$$h_i = E(K_0, m'_i \otimes i) \text{ for } i = 1, 2, \dots, s$$

Donde K_0 es una llave conocida pública fija.

Para que el receptor en el otro extremo obtenga el K_0 , el cual es la llave para el uso de **AONT**, el receptor realiza el siguiente cálculo:

$$K' = m'_s \otimes h_1 \otimes h_2 \otimes \dots \otimes h_s$$

$$m_i = m'_i \otimes E(K', i) \text{ for } i = 1, 2, \dots, s$$

AONT toma el mensaje de texto sin formato de entrada y los transforma, luego crea un bloque para almacenar los mensajes transformados antes de pasar al proceso de autentificación. Después, se genera el paquete Chaff (la cantidad de paquetes Chaff no tiene que ser igual a los paquetes de la información útil).

Esta técnica produce una menor sobrecarga que la sugerencia número 1. El AONT ofrece más confidencialidad al esquema de Chaffing and Winnowing, ya que el adversario debe recibir todo el bloque de mensajes de transformación e identificar correctamente todo el paquete de la información útil para obtener el mensaje de texto original. La Figura 2.11, muestra la descripción general de Chaffing y Winnowing si se agrega la función AONT [18].

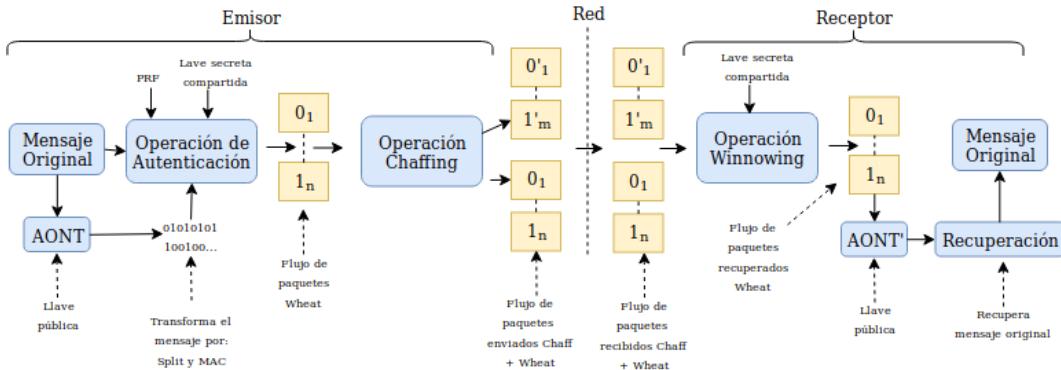


Figura 2.11: Proceso de Chaffing and Winnowing junto con AONT.

¿Cómo AONT puede hacer la diferencia?

1. Requiere menos ancho de banda al transferir paquetes, ya que se pueden transferir más bits en un paquete en lugar de un bit por paquete.
2. Los paquetes Chaff son más fáciles de generar, ya que AONT transforma el mensaje de texto plano en bits aleatorios.
3. La distinción entre Chaffing and Winnowing es más difícil: Si el adversario va a ejercer fuerza bruta en los paquetes, la tarea se ralentizará por el factor del número de bloque de mensajes. Dado que el bloque de mensaje de información adicional se mezcla aleatoriamente dentro de los flujos de paquetes de Chaffing and Winnowing, sin saber que es muy difícil que el bloque adicional proporcione la posibilidad de elegir el bloque de mensaje correcto de los paquetes para obtener el texto plano original.

2.6.7. Comparando Chaffing and Winnowing contra Cifrado y Esteganografía

En esta sección explicaremos porque Chaffing and Winnowing no puede ser clasificado como una técnica de cifrado o Esteganografía.

2.6.7.1. Chaffing and Winnowing vs Cifrado

Nosotros podríamos clasificar Chaffing and Winnowing como un método de cifrado, pero volvamos a recordar el principio de un Cifrado. El principal

objetivo de un cifrado es ocultar el mensaje en texto plano de tal manera que oculta su contenido con el uso de una clave de cifrado para el texto cifrado. Por otro lado, en el esquema original de Chaffing and Winnowing, una llave compartida es usada con el fin de autenticar la validación de los paquetes ya sea del emisor o del receptor. Además, Chaffing and Winnowing no hace uso de ninguna técnica de cifrado para ocultar el contenido de un mensaje y que nadie pueda ver dicho mensaje, solo aquellos con la llave correspondiente pueden determinar que paquetes contienen la información valida. En la figura 2.11, se muestra como se puede ver el esquema Chaffing and Winnowing como una técnica de cifrado.

Chaffing y Winnowing pueden verse como **un tipo especial de esquema de cifrado simétrico**, ya que la operación **chaffing** es similar al "proceso de cifrado". En la operación de chaffing, el texto cifrado se crea para producir un paquete de la información útil no válido que se envía al receptor. Luego, el receptor realiza el "proceso de descifrado", que implica descartar el paquete de desperdicios y recuperar el mensaje original. Ambas operaciones operan bajo una llave secreta común que se usa para derivar el valor MAC.

Pero la diferencia es, *Chaffing y Winnowing* dos partes que no buscan lograr la confidencialidad. El emisor comparte una clave secreta con el receptor para que el receptor pueda usar la clave secreta para autenticarse (si se afirma que el mensaje recibido proviene del remitente deseado). Pero la ganancia de confidencialidad proviene de la dificultad de distinguir el paquete Chaff del paquete de la información útil. Mientras que en el cifrado, la clave se utiliza para lograr la confidencialidad mediante la creación de texto cifrado que oculta el contenido del mensaje de personas [25].

Chaffing y Winnowing junto con el esquema AONT, el esquema en sí es muy parecido al cifrado, excepto que la clave que se usa en la transformación AONT, se elige aleatoriamente cada vez en lugar de fijarla. Además, el último bloque de mensajes es exclusivo o de la clave y todo el hash del bloque de mensajes está allí para garantizar que cualquier modificación en el bloque de mensajes cambiará la clave K' calculado por el receptor. Por lo tanto, el último bloque de mensajes $m'_{s'}$ está allí solo con el propósito de autenticación. Por lo tanto, Chaffing y Winnowing con el esquema AONT no pueden ser clasificados bajo cifrado [18].

2.6.7.2. Chaffing and Winnowing vs Esteganografía

La esteganografía trata de cómo ocultar, dentro de un mensaje público, información secreta. La agregación de información 'extra' es también un secreto, es decir, nadie externo a la comunicación sabe que en ese mensaje público se ha agregado información [19]. Un ejemplo de esta técnica de cifrado, se puede visualizar claramente en la utilización de tinta invisible en cartas. El emisor escribe una carta común y corriente con tinta visible, pero además, escribe información secreta con tinta invisible. Luego entonces, el receptor, aplicando el método correspondiente, descubrirá la información mandada con tinta invisible, pero agentes externos, ni siquiera saben la existencia de dicha información. En la actualidad, la esteganografía es muy utilizada para ocultar información en archivos de imágenes.

Para algunas personas, Chaffing and Winnowing puede ser clasificado como una técnica esteganografía. Sin embargo, el objetivo principal de la estenografía, como se mencionó antes, es el de ocultar el mensaje original dentro de otro mensaje, por lo tanto, nadie aparte del emisor y el receptor sabrá que hay un mensaje oculto. Contrario a Chaffing and Winnowing, en donde cualquiera puede ver el contenido del mensaje, ya que este método no trata de esconderlo de los posibles atacantes. Otra diferencia es que en esteganografía el emisor tiene que ocultar el mensaje el mismo, mientras que en Chaffing and Winnowing no necesariamente es así, ya que una "tercera parte" puede hacerlo [20].

Por lo tanto, Chaffing and Winnowing no puede ser considerado como esteganografía.

Para nuestro trabajo terminal usaremos *Chaffing and Winnowing*, proponiendo así un nuevo método de autentificación para servicios web.

Prototipo 1.

Capítulo 3

Análisis.

3.1. Estudio de Factibilidad.

El estudio de factibilidad es un instrumento que sirve para orientar la toma de decisiones en la evaluación de un proyecto y corresponde a la última fase de la etapa pre-operativa dentro del ciclo del proyecto. Se formula con base en información que tiene la menor incertidumbre posible para medir las posibilidades de éxito o fracaso de un proyecto, apoyándose en él se tomará la decisión de proceder o no con su implementación. Este estudio establecerá la viabilidad, si existe, del trabajo.

- Factibilidad Técnica: Hace referencia a los recursos como herramientas, conocimientos, habilidades, experiencia, etc. que son necesarios para efectuar las actividades del trabajo terminal.
- Factibilidad Operativa: Se refiere a los recursos necesarios para llevar a cabo los procesos de forma eficiente , depende de los recursos humanos.
- Factibilidad Económica: Consiste en los recursos financieros necesarios para llevar a cabo la elaboración de este trabajo.

3.1.1. Factibilidad Técnica

En esta parte explicaremos detalladamente las tecnologías que usaremos. Para la elección de estas herramientas fue necesario investigar las tecnologías que más se usan en la actualidad, además de ver las características y equipos de cómputo con los que contamos actualmente.

Factibilidad Técnica	
Sistema Operativo	Multiplataforma
Navegador Web	Google Chrome
Lenguaje de Programación	JavaScript
Servidor	Apache 2.0

Cuadro 3.1: Herramientas de Software a utilizar

Además de las herramientas de software a utilizar, es necesario mencionar el equipo de hardware que utilizaremos tanto para desarrollar como para probar e implementar cada uno de los prototipos que se mencionarán a lo largo de este trabajo terminar, el cual es:

Equipo de hardware [1]	
Marca	DELL
Modelo	Inspiron 5567
Procesador	Intel Core i7 7gen
Tarjeta de video	Radeon (TM) R7 M445
Memoria RAM	16 GB
Disco Duro	256 GB SSD y 1 TB HDD

Cuadro 3.2: Equipo de hardware a utilizar [1]

Equipo de hardware [2]	
Marca	Asus
Modelo	FX505GM-BN061T
Procesador	Intel Core i5 8th Gen
Tarjeta de video	NVidia GeForce GTX 1060
Memoria RAM	8 GB
Disco Duro	256 GB SSD y 1 TB HDD

Cuadro 3.3: Equipo de hardware a utilizar [2]

Equipo de hardware [3]	
Marca	HP
Modelo	Pavilion g4
Procesador	Intel Core i3
Tarjeta de video	Intel Sandybridge Mobile
Memoria RAM	6 GB
Disco Duro	500 GB

Cuadro 3.4: Equipo de hardware a utilizar [3]

Junto con las herramientas de hardware y software a utilizar es necesario mencionar una serie de servicios básicos que son relevantes para el desarrollo de este trabajo terminal como lo son:

- Luz Eléctrica
- Agua Potable
- Internet
- Papelería en general

Estos servicios forman parte de la factibilidad técnica ya que sin ellos no se podría realizar este trabajo terminal y por eso mismo generan un costo, dicho costo se menciona en la Factibilidad Económica.

3.1.2. Factibilidad Operativa

Los recursos operativos de este trabajo terminal se calcularon con base en los recursos humanos con los que se cuenta y un análisis de las horas que el personal estará en operación trabajando sobre éste, el cual se muestra a continuación:

Horas a trabajar en el desarrollo del trabajo terminal						
Mes	No. de Días	Sábado y Domingo	Días hábiles	Horas de trabajo por día	Horas Totales	Días laborables (8 hr.)
Enero	31	8	9	2	18	2
Febrero	28	8	19	2	38	4
Marzo	31	10	20	2	40	5
Abril	30	10	15	2	30	3
Mayo	31	8	18	2	36	4
Junio	30	10	8	2	16	2
Agosto	31	9	12	2	24	3
Septiembre	30	10	16	2	32	4
Octubre	31	10	20	2	40	5
Noviembre	31	8	18	2	36	4

Cuadro 3.5: Relación de horas de trabajo estimadas para la realización de este trabajo terminal

Con esto podemos concluir que contamos suficiente tiempo para el desarrollo de este trabajo terminal, ya que las horas totales de trabajo están contempladas para cada uno de los integrantes del equipo

3.1.3. Factibilidad Económica

Luego de haber realizado el estudio de factibilidad técnica así como el operacional es necesario tomar en cuenta un estudio de factibilidad económica el cuan desglosará todo el gasto económico realizado para la elaboración de este trabajo terminal:

- Capital Humano: Se tienen contemplados aproximadamente 36 días laborales, es decir 288 horas para la elaboración de este trabajo terminal en el cual participaremos los cuatro integrantes
- Capital Técnico: Se cuentan con las instalaciones de la escuela, así como las viviendas de cada uno de los integrantes y los equipos de cómputo correspondientes.

En cuanto a los costos monetarios de todo el trabajo terminal se tiene lo siguiente:

- Servicios
En cuando a los servicios se considera un gasto mensual aproximado

de \$1,600.00 que al multiplicarlo por todo el tiempo de elaboración tenemos \$ 16,000.00.

- Software

En este caso durante todo el trabajo terminal usaremos herramientas gratuitas y la mayoría de software libre por lo que no dedicaremos una parte monetaria en el gasto de este tipo.

- Hardware

En este caso y como se mencionó anteriormente utilizaremos los equipos de cómputo personales de cada integrante lo que da un costo aproximado total de \$ 35,000.00.

- Recursos Humanos

Estamos estimando un gasto de \$80,000.00 por cada integrante para la elaboración de este trabajo terminal por lo que se genera un gasto total de \$320,000.00

Por lo que el costo final del desarrollo de este trabajo terminal es:

\$371,000.00

Conclusión Tras analizar todo este trabajo terminal y cada una de las partes del estudio de factibilidad es pertinente decir que los integrantes no contarán con el apoyo financiero antes mencionado y que el hardware actualmente ya es propiedad de los integrantes, por lo que el trabajo terminal se califica como "Viable" iniciando de esta manera su implementación acorde con las fechas mencionadas.

3.2. Herramientas a usar.

3.2.1. Software.

Para el desarrollo de software de este prototipo, es necesario hacer mención de algunas de las siguientes herramientas, para tener una idea clara sobre qué herramientas estamos utilizando y porque es que las estamos utilizando:

HTML5. HTML comenzó mucho tiempo atrás con una simple versión propuesta para crear la estructura básica de páginas web, organizar su contenido y compartir información, todo esto tenía la intención de comunicar información por medio de texto. El limitado objetivo de html motivó a varias

compañías a desarrollar nuevos lenguajes y programas para agregar características a la web nunca antes implementadas.

Dos de las opciones propuestas fueron Java y Flash; ambas fueron muy aceptadas y consideradas como el objetivo de la internet, sin embargo, con el crecimiento exponencial del internet, éste dejó de ser únicamente para los aficionados de los computadores y pasó a ser usado como un campo estratégico para los negocios y para la interacción social, ciertas limitaciones presentes en ambas tecnologías probaron ser una sentencia de muerte. Esta falta de integración resultó ser crítica y preparó el camino para la evaluación de un lenguaje del cual hablaremos un poco más a detalle después: JavaScript. Sin embargo, pese a su gran impacto, el mercado no terminó de adoptarlo plenamente y rápidamente su popularidad fue declinando, y el mercado terminó enfocando su atención a Flash. No fue hasta que los navegadores mejoraron su intérprete para JavaScript y la gente se empezaba a dar cuenta de las limitaciones que ofrecía Flash, que JavaScript fue implementado y comenzó a innovar la forma en la que se programaba la web. Al cabo de unos años, JavaScript, html y css eran considerados como la más perfecta combinación para evolucionar la Web.

HTML5 es una mejora de esta combinación, lo que unió todos estos elementos. HTML5 propone estándares para cada aspecto de la Web y también un propósito claro para cada una de las tecnologías involucradas. A partir de esto, html provee los elementos estructurales, CSS se concentra en volver esta estructura utilizable y atractiva a la vista, y JavaScript tiene todo lo necesario para brindar dinamismo y construir aplicaciones web completamente funcionales. Cabe mencionar que HTML5 funciona diferente dependiendo del navegador y la versión en la que se esté trabajando, algunos soportan más características o diferentes funcionalidades que otros.

CSS3.

Ya se ha mencionado anteriormente como es que HTML5 fue evolucionando a un grado de combinación de estructura y diseño, sin embargo, la web demanda diseño y funcionalidad, no solamente organización estructural o definición de secciones, la función de CSS se concentra en volver la estructura de HTML utilizable y atractivo a la vista.

Oficialmente CSS no tiene nada que ver con HTML4, no es parte de la especificación, es de hecho, un complemento desarrollado para superar las limitaciones y reducir la complejidad de HTML. Al principio, atributos den-

tro de las etiquetas HTML proveían estilos esenciales para cada elemento, pero a medida que HTML evolucionó, la escritura de códigos se volvió más compleja y HTML por sí mismo no pudo satisfacer más las demandas de los diseñadores. En consecuencia a esta demanda, CSS fue adoptado como la forma de separar la estructura de la presentación, y ha ido creciendo y ganando importancia, pero siempre desarrollado en paralelo enfocado en las necesidades de los diseñadores y apartado de la estructura de HTML.

La versión 3 de CSS sigue el mismo camino, pero esta vez con un mayor compromiso. La especificación de HTML5 fue desarrollada considerando CSS a cargo del diseño. Debido a esta consideración, la integración entre HTML y CSS es ahora vital para el desarrollo web y esta razón por la que cada vez que mencionamos HTML5 también estamos haciendo referencia a CSS3, aunque oficialmente se trate de dos tecnologías completamente separadas. Las nuevas características incorporadas en CSS3 están siendo implementadas e incluidas junto al resto de la especificación en navegadores compatibles con HTML5 [10].

Frameworks CSS.

Un framework de CSS es una biblioteca de estilos genéricos que puede ser usada para implementar diseños web. Aportan una serie de utilidades que pueden ser aprovechadas frecuentemente en los distintos diseños web. Un framework de CSS, si está bien diseñado e implementado, proporciona las siguientes ventajas [35]:

- Proporcionar una forma fácil y por tanto rápida de implementar diseños web.
- Nos aseguran que el diseño va a funcionar en una amplia gama de navegadores.
- Nos aseguran que su código cumple cierta norma estándar.
- Nos aseguran cierto grado de fiabilidad en la eficacia de las utilidades que nos aportan. El framework se supone que está bien probado para asegurarnos que no hay errores.

Sin embargo, un framework de CSS puede llevar aparejado las siguientes desventajas[35]:

- La importación de código del framework que no es necesario en nuestro diseño web concreto. Esto provoca un incremento innecesario del consumo del ancho de banda y del tiempo de descarga.

- Hay un menor control por parte del maquetador de lo que realmente está sucediendo en la visualización de la página web. Esto suele ser un problema cuando se tiene que corregir algún efecto indeseado.
- Al diseñar con código prehecho, podemos estar limitándonos en cuanto las posibilidades de elección del diseño web.

Bootstrap.

Bootstrap es framework de código abierto que contiene HTML, CSS y JS, en la actualidad, Bootstrap se ha convertido en uno de los frameworks de front-end más importantes en la actualidad. [36]

Materialize.

Materialize es un framework CSS creado y diseñado por Google. Combina los principios clásicos de diseño con las tecnologías e innovación moderna de Material Design. El objetivo de materialize es desarrollar un sistema de diseño que permita la unificación de la experiencia del usuario a través de los productos de google.

JavaScript.

JavaScript es considerado como el lenguaje de programación de html y de la web. Es un lenguaje de programación fácil de usar y muy versátil para el ámbito de la comunicación en redes. Los programas, llamados "scripts", se ejecutan en el navegador (Mozilla, Google Chrome, Internet Explorer, etc.) normalmente consisten en unas funciones que son llamadas desde el propio html cuando algún evento sucede.

Su primera aproximación a un uso real, fue en mayor parte para "dar vida a una página web", como dar animaciones a un botón, interacciones en tiempo real, entre otras más. JavaScript fue desarrollado por Netcape, a partir del lenguaje Java, que en ese momento tenía mucho auge y popularidad, y su principal diferencia es que JavaScript sólo "funciona" dentro de una página html.

JavaScript fue declarado como estándar del European Computer Manufacturers Association (ECMA) en 1997, y poco después, también fue estandarizado por ISO [22].

JavaScript es un lenguaje interpretado, usado mayormente como complemento de ciertos objetivos específicos, sin embargo, uno de las innovaciones que ayudó a JavaScript fue el desarrollo de nuevos motores de interpretación, creados para acelerar el procesamiento del código. La clave de los motores más exitosos fue transformar el código de Javascript en código máquina para

obtener una velocidad de ejecución mejor que antes. Esto a la vez permitió superar viejas limitaciones de rendimiento y confirmar el lenguaje JavaScript como la mejor opción para la Web.

Para aprovechar esta prometedora plataforma de trabajo ofrecida por los nuevos navegadores, JavaScript fue expandido en cuestión de portabilidad e integración, a la vez, interfaces de programación de aplicaciones (APIs) fueron incorporando por defecto con cada navegador para asistir a JavaScript en funciones elementales. El objetivo de esto, fue principalmente hacer disponible funciones a través de técnicas de programación sencillas y estándares, expandiendo el alcance del lenguaje y facilitando la creación de programas útiles para la Web [10].

JQuery.

Antes de continuar con JQuery, debemos saber que es un framework. Un framework es un producto que sirve como base para la programación avanzada de aplicaciones, la cual aporta una serie de funciones o códigos para realizar tareas habituales. Dicho de otra manera, un framework son librerías de código que contienen procesos o rutinas listos para hacer uso. Habitualmente los programadores utilizan los frameworks para no tener que desarrollar ellos mismos las tareas más básicas, puesto que en el propio framework ya hay implementaciones que están probadas, funcionan y no se necesitan volver a programar.

Una vez comprendido que es un framework podemos continuar con jQuery. Este framework (para el lenguaje Javascript), es un producto que nos simplifica la vida para programar en este lenguaje. jQuery implementa una serie de clases (programación orientada a objetos) que nos permiten programar sin preocuparnos del navegador con el que nos está visitando el usuario, ya que funcionan de forma exacta en todas las plataformas más habituales. Así pues, este framework Javascript, nos ofrece una infraestructura con la que tendremos mucha mayor facilidad para la creación de aplicaciones complejas del lado del cliente. Por ejemplo, con jQuery obtendremos ayuda en la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de Ajax, etc. Cuando se programa Javascript con jQuery se tiene a su disposición una interfaz para programación que nos permitirá hacer cosas con el navegador que estemos seguros que funcionarán para todos los visitantes de la página.[12]

CryptoJs.

CryptoJs es una colección estándar y segura de algoritmos criptográficos implementados para JavaScript usando las mejores prácticas y patrones. En este trabajo usaremos CryptoJS en el desarrollo del Componente I, específicamente, utilizaremos los algoritmos de cifrado SHA-256 y AES, ya implementados con ella.[13]

NodeJs.

NodeJS es un framework de ejecución de JavaScript orientado a eventos asíncronos para construir aplicaciones en red escalables. Cabe destacar que a diferencia de la mayoría del código JavaScript, no se ejecuta en el navegador, sino en un servidor. [37]

NodeJs tiene una amplia cantidad de librerías para el desarrollo web, una herramienta de la cual nosotros haremos uso es la librería ó paquete de OpenSSL, el cual nos permite crear certificados SSL (x509) con código más limpio, ya que este paquete se encarga de hacer las llamadas al sistema para la creación de los certificados, haciendo llamadas a sus funciones nos permite crear ó revocar, entre muchas otras acciones, las cuales nos ayudarán a la creación del componente 2 (Autoridad certificadora). [38]

Java.

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. El lenguaje de programación Java fue originalmente desarrollado por James Gosling, de Sun Microsystems (constituida en 1983 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente [39].

Spring Framework.

Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web sobre la plataforma Java EE. A pesar de que no impone ningún modelo de programación en particular, este framework

se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (Enterprise JavaBean) [40]. En este trabajo utilizaremos Spring Framework 5 para el desarrollo del Componente III, específicamente la API.

Apache Tomcat.

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por Sun Microsystems).

Tomcat es un contenedor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java [41].

En este trabajo terminal, estaremos usando Apache Tomcat 9.

OpenSSL.

OpenSSL es un proyecto de software libre basado en SSLeay, desarrollado por Eric Young y Tim Hudson.

Consiste en un robusto paquete de herramientas de administración y bibliotecas relacionadas con la criptografía, que suministran funciones criptográficas a otros paquetes como OpenSSH y navegadores web (para acceso seguro a sitios HTTPS). Estas herramientas ayudan al sistema a implementar el Secure Sockets Layer (SSL), así como otros protocolos relacionados con la seguridad, como el Transport Layer Security (TLS). OpenSSL también permite crear certificados digitales que pueden aplicarse a un servidor, por ejemplo Apache [29].

Para este trabajo terminal utilizaremos OpenSSL para generar los certificados de los usuarios, así como los certificados autofirmados para el Componente II y el Componente III.

MongoDB.

MongoDB es un sistema de base de datos multiplataforma orientado a documentos, de esquema libre, esto significa que cada entrada o registro puede tener un esquema de datos diferente, con atributos o "columnas" que no tienen por qué repetirse de un registro a otro.

Las características más destacadas son su velocidad y su sencillo sistema de consulta de los contenidos de la base de datos. Alcanzando así un balance perfecto entre rendimiento y funcionalidad. MongoDB utiliza un modelo NoSQL el cual es un modelo de agregación que se basan en la noción de agregado, entendiendo el agregado como una colección de objetos relacionados que se desean tratar de forma semántica e independiente [28].

Las ventajas que ofrece MongoDB como herramienta de desarrollo de base de datos no relacionales son:

- La base de datos no tiene un esquema de datos predefinido.
- El esquema puede variar para instancias de datos que pertenecen a una misma entidad.
- En ocasiones el gestor de la base de datos no es consciente del esquema de la base de datos.
- Permite reducir los problemas de concordancia entre estructuras de datos usadas por las aplicaciones y la base de datos.
- Frecuentemente se aplican técnicas de desnormalización de los datos.

MySQL.

Es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web [42].

El modelo relacional, para el modelado y la gestión de bases de datos, es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente [43].

Las principales ventajas son:

- Provee herramientas que garantizan evitar la duplicidad de registros.
- Garantiza la integridad referencial, así, al eliminar un registro elimina todos los registros relacionados dependientes.
- Favorece la normalización por ser más comprensible y aplicable.

Mientras que las principales desventajas son:

- Presentan deficiencias con datos gráficos, multimedia, CAD y sistemas de información geográfica.
- No se manipulan de forma eficiente los bloques de texto como tipo de dato.

JSEncrypt.

Generada por Travis Tidwell, JSEncrypt es una librería para JavaScript que permite el cifrado, descifrado y la generación de llaves de RSA con OpenSSL.

VSFTPD.

VSFTPD son las siglas para Very Secure File Transport Protocol Daemon y es un servidor de FTP que nos da la ventaja de garantizarnos seguridad, estabilidad y desempeño al momento de montarlo, además de que nos permite configurar su control de acceso mediante una serie de criterios, un factor muy útil y necesario que utilizaremos más adelante para el desarrollo de nuestro segundo prototipo. [45] La herramienta se escogió debido a la comparación de los diferentes protocolos de internet que permiten el intercambio de archivos, los cuales son:

- **FTP:** *File Transfer Protocol* ó *Protocolo de transferencia de archivos* es un protocolo para transferencia de archivos entre sistemas conectados a una red. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo. FTP nos permite tener un control de accesos de las diferentes carpetas, dando así diferentes prioridades. Al subir archivos al servidor FTP si se remplaza alguno, éste no podrá recuperarse [47].
- **HTTP:** *Hypertext Transfer Protocol* ó *Transferencia de HiperTexto* es un protocolo cliente-servidor que establece los intercambios de información entre los clientes web y los servidores HTTP. La versión mas

reciente de HTTP es la 1.1, y su especificación se encuentra recogida en el documento RFC 2616. HTTP se establece sobre la capa TCP/IP [48]. Las tres operaciones más usadas que permiten a un cliente dialogar con el servidor son [47]:

1. GET: Recoger un objeto
2. POST: Enviar información al servidor
3. HEAD: Solicitar las características de un objeto

- **TFTP:** *Protocolo Trivial de Transferencia de Archivos* es un protocolo que proporciona la transferencia de archivos sin autenticación de usuario. TFTP está más enfocado a las aplicaciones que no necesitan las interacciones sofisticadas que proporciona el protocolo de transferencia de archivos (FTP) [49].
- **SMTP:** *Simple Mail Transfer Protocol* ó *Protocolo Simple de Transferencia de Correo* es protocolo que funciona en línea, encapsulado en una trama TCP/IP. El correo se envía directamente al Servidor de Correo del destinatario. El protocolo SMTP funciona con comandos de textos enviados al servidor SMTP. A cada comando enviado por el Cliente le sigue una respuesta del servidor SMTP compuesta por un número y un mensaje descriptivo [50].
- **HTTPS:** *HyperText Transfer Protocol Secure* ó *Protocolo Seguro de Transferencia de Hipertexto* es un protocolo de comunicación de Internet que protege la integridad y la confidencialidad de los datos de los usuarios entre sus ordenadores y el sitio web. Los datos que se envían mediante HTTPS están protegidos con el protocolo Seguridad en la capa de transporte (TLS), que da estas tres capas clave de seguridad [51]:
 1. Cifrado: Se cifran los datos intercambiados.
 2. Integridad de los datos: Los datos no pueden modificarse ni dañarse durante las transferencias.
 3. Autenticación: Demuestra que los usuarios se comunican con el sitio web previsto. Proporciona protección frente a los ataques *man-in-the-middle*.

Wireshark.

Wireshark es una analizadora de paquetes de red. Un analizador captura paquetes de red y muestra los datos del paquete con el mayor detalle posible.

Ésta herramienta es software libre y es el mejor analizador de paquetes hoy en día [30]. Utilizaremos Wireshark para analizar los paquetes que viajan a través de la red, en este caso la petición modificada la cual contiene el certificado y el patrón ocultados mediante el método *Chaffing*.

3.2.2. Hardware.

En el ámbito del hardware, utilizaremos los equipos de cómputo con los cuales contamos actualmente los integrantes de este equipo, los cuales se especificarán a continuación:

Equipo de hardware utilizado. [1]	
Marca	Asus
Modelo	FX505GM-BN061T
Procesador	Intel Core i5 8th Gen
Tarjeta de video	NVidia GeForce GTX 1060
Memoria RAM	8 GB
Disco duro	256 GB SSD y 1 TB HDD

Equipo de hardware utilizado. [2]	
Marca	HP
Modelo	Pavilion g4
Procesador	Intel Core i3
Tarjeta de video	Intel Sandybridge Mobile
Memoria RAM	6 GB
Disco duro	500GB

Equipo de hardware utilizado. [3]	
Marca	DELL
Modelo	Inspiron 5567
Procesador	Intel Core i7
Tarjeta de video	Radeon (TM) R7 M445
Memoria RAM	16 GB
Disco duro	256 GB SSD y 1 TB HDD

Equipo de hardware utilizado. [4]	
Marca	HP
Modelo	Pavilion 15-p000ns
Procesador	AMD A A8-5545M
Tarjeta de video	Radeon R8 M540
Memoria RAM	8 GB
Disco duro	1TB

3.3. Arquitectura del sistema.

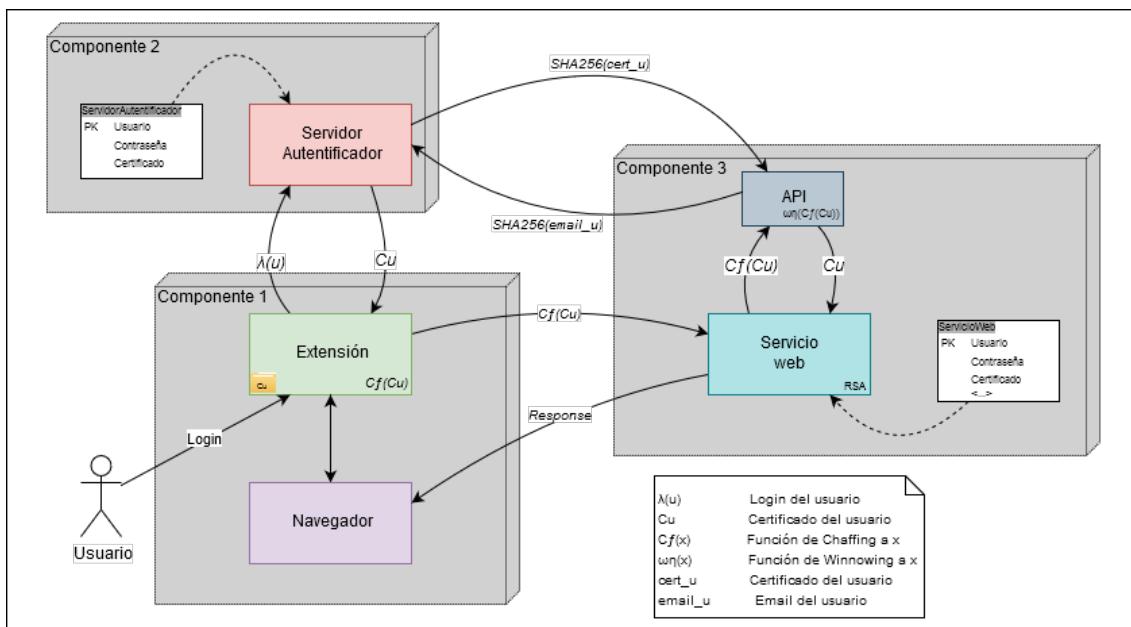


Figura 3.1: Arquitectura General del Sistema

3.3.1. Descripción de la arquitectura del sistema.

El sistema se compone de 3 grandes bloques los cuales se comunicarán vía red:

1. **Navegador Chrome con la Extensión instalada:** Este primer bloque es el que se encuentra interactuando directamente con el usuario de nuestro sistema, consiste en la extensión creada por nosotros y el

navegador en el que el usuario realiza peticiones a diferentes servicios en la web.

2. **Servidor autentificador:** Este bloque va ser el encargado de generar los certificados para cada usuario que se registre en la extensión y enviarlos a la extensión. Para la generación de dichos certificados utilizaremos una autoridad certificadora con lo que garantizamos la seguridad de estos mismos. Por otro lado para almacenar los datos de nuestros usuarios contaremos con una tabla que contenga como principales campos:

- Usuario
- Contraseña
- Certificado

3. **Servidor web con API instalada:** En este módulo el servicio web contará con una API, que se encargará de reconocer las peticiones que se reciban con nuestro método de autenticación y será la encargada de interpretar los datos y facilitarle la información de autenticación al servicio. Es importante destacar que el servicio almacenará el certificado en cuestión para que el usuario pueda autenticarse la próxima vez de forma automática.

Es importante mencionar que la comunicación entre cada uno de los bloques se realizará mediante técnicas que permitan la confidencialidad de los datos, por un lado la comunicación del certificado que viajará entre la extensión y el servicio web se encontrará oculto mediante Chaffing and Winnowing y el patrón necesario para el método se encontrará cifrado mediante RSA. La comunicación entre el Servidor autentificador y la Extensión se encontrará oculto mediante un socket seguro.

3.4. Diagrama BPMN

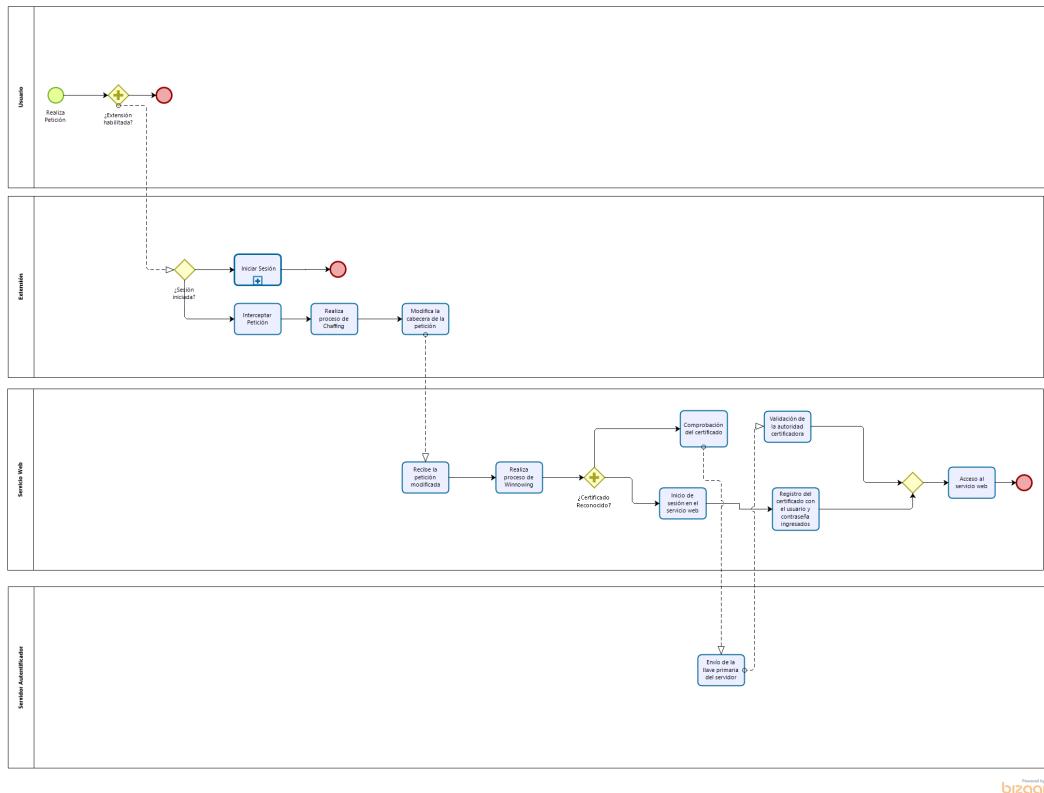


Figura 3.2: Diagrama de Modelo y Notación de Procesos de Negocio

3.4.1. Diagrama BPMN Proceso: Inicio de Sesión

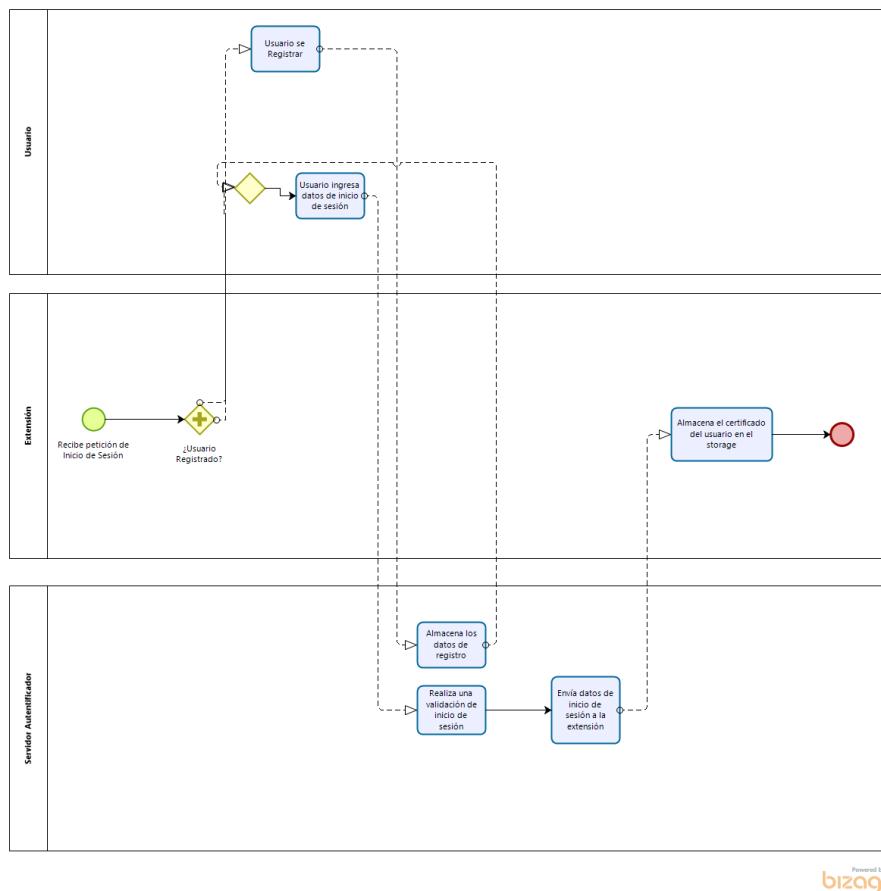


Figura 3.3: Diagrama BPMN del proceso Iniciar Sesión

Powered by
bizagi
Modeler

3.5. Diagrama de casos de uso general.

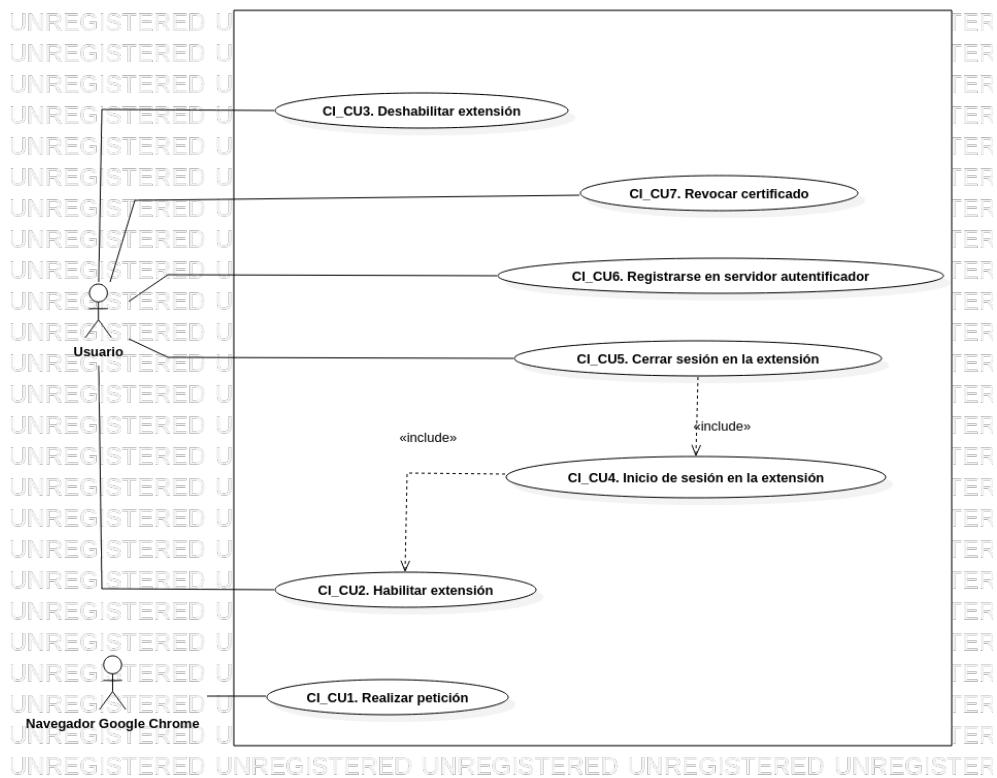


Figura 3.4: Diagrama de casos de uso general del sistema

NOTA: En la sección 4 "Diseño", cada prototipo describirá sus casos de uso correspondientes para su funcionamiento.

3.6. Componente I. Extensión.

3.6.1. Descripción.

Este componente permite a la extensión poder interceptar peticiones hechas por el usuario a través del navegador de Google Chrome.

Una vez que se intercepta la petición, ésta podrá ser modificada. La modificación se hará sólo mientras la extensión esté habilitada, y tiene como objetivo injectar el certificado autenticador en el encabezado del protocolo una vez que se haya llevado a cabo el proceso de Chaffing. Cuando dicho certificado

sea injectado, la extensión deberá liberar la petición para que salga a red. Este certificado será único por cada usuario y será obtenido desde el Componente II cuando el usuario inicie sesión en la extensión. Si el usuario no puede iniciar sesión debido a que no tiene una cuenta, desde la extensión se podrá registrar para poder obtener su certificado.

Además, el usuario podrá cerrar la sesión en la extensión si es que así lo desea, lo que eliminaría el certificado de la máquina local del usuario. Por otro lado, si el usuario desea revocar su certificado, lo podrá hacer también desde este componente.

El propósito de este prototipo es utilizar la técnica de *Chaffing and Winnowing* en este nuevo método de autentificación, para evitarle al usuario la tediosa tarea de ingresar sus credenciales cada vez que accede al servicio y brindarle la seguridad necesaria al iniciar sesión.

Para injectar el certificado autentificador, es necesario crear un "*patrón de chaffing*", este patrón lo generaremos aleatoriamente para después mandarlo junto con la petición HTTP. Dicho patrón irá cifrado con la clave pública del Componente III (API). El objetivo de mandar el patrón junto con el protocolo HTTP, es que el servidor pueda descifrar el patrón con su clave privada y con él realizar la etapa de *winnowing* para extraer el certificado.

3.6.2. Estudio de requerimientos.

3.6.2.1. Requerimientos Funcionales.

CI_RF1. Interceptar petición HTTP. La extensión deberá interceptar la petición HTTP del navegador, en cuanto el usuario realice alguna a través de éste.

CI_RF2. Deshabilitar extensión. El usuario podrá deshabilitar la extensión, para que ésta no vigile su actividad en el navegador.

CI_RF3. Habilitar extensión. El usuario podrá habilitar la extensión, para que ésta vigile las peticiones HTTP.

CI_RF4. Validar petición. La extensión deberá analizar la petición previamente interceptada, y validar si ésta es HTTP(S) o no.

CI_RF5. Interceptar petición. La extensión deberá evitar que la peti-

ción salga a red, deteniéndola para aplicar la etapa de *Chaffing*.

CI_RF6. Inicio de sesión en la extensión. La extensión contará con una interfaz para el ingreso de datos del usuario, donde ingresará un usuario y contraseña.

CI_RF7. Obtención del certificado autentificador. La extensión, mediante el inicio de sesión del usuario, se conectará al Componente II (Servidor autentificador) para obtener el certificado autentificador.

CI_RF8. Almacenamiento del certificado autentificador. La extensión deberá almacenar el certificado autentificador devuelto por la autoridad certificadora.

CI_RF9. Generación de patrón de Chaffing. La extensión generará un patrón para poder implementar el método de *Chaffing*. Este patrón será generado al azar, y es aquel que se usará para introducir el código autentificador en el protocolo HTTP.

CI_RF10. Etapa de Chaffing. Por medio del método *Chaffing* se agregará al encabezado HTTP el certificado del usuario, utilizando el patrón de *Chaffing* del requerimiento funcional CI_RF9. Generación de patrón de *Chaffing*

CI_RF11. Liberación de Petición. Se liberará el bloqueo a la petición HTTP impuesto por el requerimiento funcional CI_RF5. Interceptar petición.

CI_RF12. Cierre de sesión en la extensión. El usuario podrá cerrar sesión en la extensión para que ésta no siga guardando su certificado.

CI_RF13. Registro en servidor autentificador. La extensión contará con una interfaz para el ingreso de datos del usuario, para que éste se registre en el servidor y pueda obtener un certificado. Los datos a ingresar serán: correo electrónico (email), y contraseña.

CI_RF14. Revocar certificado. El usuario podrá, mediante la interfaz de la extensión, revocar su certificado en caso de que así lo deseé. La interfaz requerirá que se introduzca el email correspondiente al usuario y la contraseña para validar la identidad del usuario.

CI_RF15. Ver contraseña. Si el usuario así lo desea, podrá habilitar

la función ver contraseña cuando escribe. Este requerimiento se usara en los requerimientos CI_RF6. Inicio de sesión en la extensión y CI_RF13. Registro en servidor autentificador.

3.6.2.2. Requerimientos no Funcionales.

CI_RNF1. Plataforma de implementación. La extensión será implementada en el navegador Google Chrome Desktop.

CI_RNF2. Versión del navegador La extensión funcionará a partir de la versión 28.0.

CI_RNF3. Tecnologías para la interfaz de usuario Para el sistema se hará uso de HTML5, JavaScript, CSS3, JSON.

CI_RNF4. Codificación en base64 del Chaffing. Se codificará en base64 la cadena de chaffing para permitir el envío de los datos, debido a que el algoritmo nos devuelve ciertos caracteres no permitidos para enviar.

CI_RNF5. Conexión a internet. Para el funcionamiento de la extensión, no es necesario que se tenga conexión a internet.

CI_RNF6. Tamaño del código autentificador. El tamaño del código autentificador es de aproximadamente 10496 bits pero puede variar dependiendo de la información introducida para su elaboración.

CI_RNF7. Almacenado de archivo en la extensión. Se necesita tener almacenado el archivo en la extensión de Google Chrome. Específicamente, utilizamos el **Storage** para almacenarlo.

3.6.3. Reglas del negocio.

CI_RN1. Extensión habilitada. En cuanto el usuario lo indique por medio de la Interfaz de Usuario, la extensión deberá vigilar la actividad que éste realice en el navegador para interceptar una petición.

CI_RN2. Extensión deshabilitada. En cuanto el usuario lo indique por medio de la Interfaz de Usuario, la extensión deberá dejar de vigilar la

actividad que éste realice en el navegador.

CI_RN3. Petición válida. La extensión modificará la petición siempre y cuando se trate de una petición válida HTTP .

CI_RN4. Inicio de sesión de extensión por usuario. Cada usuario que desee utilizar la extensión sólo deberá tener una cuenta con un correo electrónico y una contraseña respectiva a este usuario.

CI_RN5. Acceso a internet. Se debe de contar con acceso a internet para que la extensión pueda enviar al servidor la petición modificada.

CI_RN6. Longitud de código autentificador. La longitud del código autentificador es de aproximadamente 10496 bits pero puede variar dependiendo de la información introducida para su elaboración.

CI_RN7. Longitud del campo de usuario. La longitud del usuario no debe pasar de 20 caracteres, y mínimo será de 3 caracteres.

CI_RN8. Longitud del campo contraseña. La longitud de la contraseña no debe pasar de 16 caracteres, y mínimo sera de 8 caracteres.

CI_RN9. Caracteres permitidos en campo usuario. Los caracteres permitidos en el campo de usuario son únicamente símbolos alfanuméricos y guion bajo.

CI_RN10. Caracteres permitidos en campo contraseña. Los caracteres permitidos en el campo de contraseña son únicamente símbolos alfanuméricos así como los símbolos _ @ / # ? .

CI_RN11. Formato de contraseña. La contraseña debe tener al menos un número y una letra mayúscula.

CI_RN12. Longitud del campo email. La longitud del email no debe pasar de 100 caracteres, y mínimo sera de 7 caracteres.

CI_RN13. Caracteres permitidos en campo email. Los caracteres permitidos en el campo email son únicamente símbolos alfanuméricos así como los símbolos ' ' '@' '_' '-' ',' '()'{}[]'.

3.7. Componente II: Servidor autentificador.

3.7.1. Descripción.

Para este componente, implementaremos un servidor autentificador en el cual se crearán y almacenarán los certificados de los usuarios. En este servidor, los usuarios tendrán registrada una cuenta a la cual accederán desde el Componente I. Una vez que el usuario inicie sesión en la extensión, este servidor autentificador regresará como respuesta el certificado generado para que la extensión lo almacene. Para la creación del certificado autentificador se utilizará la herramienta OpenSSL, además, la comunicación entre extensión y servidor se hará bajo SSL/TLS.

La principal función de este componente es gestionar las cuentas y certificados en una base de datos, de tal forma que el Componente I se pueda comunicar con este componente ya sea para que le genere un certificado a una cuenta (en el caso de que el usuario se esté registrando en el servidor autentificador por primera vez), para mandarle su certificado si es que ya se encuentra registrado en el servidor, o bien, para revocar un certificado. Así mismo, este componente también estará comunicado con el Componente III, específicamente la API, para controlar la revocación de certificados.

El propósito de este componente es poder crear y utilizar un certificado real, generado por una autoridad certificadora de confianza y con el cual se pueda autenticar a un usuario en un servicio web.

3.7.2. Estudio de requerimientos.

3.7.2.1. Requerimientos funcionales.

CII_RF1. Creación de nuevo usuario. La autoridad certificadora podrá crear una nueva instancia en la base de datos de acuerdo a los datos recuperados por la extensión (Usuario y contraseña), si es que estos no se encuentran guardados en la base de datos.

CII_RF2. Generar certificado. La autoridad certificadora deberá generar un certificado diferente para cada usuario que se registre en la extensión con los datos proporcionados por la misma.

CII_RF3. Asignar certificado. La autoridad certificadora deberá asignar el certificado generado al usuario correspondiente en la base de datos.

CII_RF4. Devolver certificado. La autoridad certificadora deberá enviar el certificado correspondiente al usuario quien realiza la solicitud para obtener el certificado.

CII_RF5. Actualizar certificado. La autoridad certificadora deberá actualizar el certificado cuanto éste haya caducado.

CII_RF6. Revocar certificado. La autoridad certificadora deberá revocar un certificado de acuerdo a la petición dada por el usuario, si esta petición tiene los datos correctos del usuario, su certificado ligado al mismo se eliminará, se creará uno nuevo y se le asignará a dicho usuario.

CII_RF7. Comprobar certificado certificado. La autoridad certificadora deberá poder comprobar la existencia de un certificado basándose en el código hash recibido por parte del Componente III.

3.7.2.2. Requerimientos no funcionales.

CII_RNF1. Plataforma de implementación. La autoridad certificadora será implementada en NodeJs.

CII_RNF2. Version de SSL/TLS. La autoridad certificadora usará como conexión, así como la generación de certificado, OpenSSL con la versión 1.1.1.

CII_RNF3. Base de datos. La autoridad certificadora se conectará a una base de datos MongoDB 4.0.10.

CII_RNF4. Formato de certificado. El servidor autenticador deberá crear certificados X509.

3.7.3. Reglas del negocio.

CII_RN1. Acceso a internet. Se debe de contar con acceso a internet para que la autoridad certificadora pueda enviar a la extensión el certificado generado.

CII_RN2. Conexión a la base de datos. La autoridad certificadora cuenta con un pull de conexiones a la base de datos para poder insertar u

obtener certificados de los usuarios.

CII_RN3. Conexión SSL. La autoridad certificadora cuenta con un certificado auto firmado, el cual le permitirá tener una conexión SSL con la extensión para poder así enviar el certificado de dicho usuario de manera segura y cifrada.

CII_RN4. Parámetros POST. La autoridad certificadora recibirá mediante método POST peticiones las cuales tendrán como datos: Correo electrónico (email) y password (Con hash 256).

CII_RN5. Parámetros GET. La autoridad certificadora enviará una pagina como respuesta si el usuario aprueba el uso del certificado de la autoridad certificadora.

CII_RN6. Respuestas a peticiones. La autoridad certificadora podrá devolver un certificado creado por ella misma. Así como también devolver códigos de respuestas, como lo son: 200 (Se realizo petición correctamente) ó 404 (No se encontró usuario).

3.8. Componente III: API.

3.8.1. Descripción.

Para el componente III, se utilizará un servicio web de prueba para mostrar la funcionalidad del inicio de sesión por este método propuesto, por lo cual dicho servicio no se ve reflejado en el análisis desarrollo del Componente III.

Se realizará la etapa de *winnowing* a la petición HTTP para obtener el certificado. Esto último lo realizará una API dedicada exclusivamente a ello, con conexión al Componente II para checar el estatus y validez del certificado.

El servicio web sólo tendrá que conectarse al API para obtener el certificado y su validez, con base en ello realizar la lógica del negocio para iniciar sesión.

Así mismo, el API implementará un cifrado asimétrico (RSA), esto con la finalidad de que el Componente I pueda tomar la llave pública del servidor y cifrar el "*patrón de chaffing*" que se mandará en la petición. Una vez que el patrón y el chaffing llegue al API, sólo ella podrá descifrar el patrón con su llave privada, dandole la integridad necesaria al patrón para viajar a través

de la red.

El propósito de este componente es poder obtener el certificado que identificará a cada usuario, para poder validarla y saber si se debe de dar o negar el acceso. Cabe mencionar que, la primera vez que el servidor reciba el certificado autenticador en el servicio, éste pedirá al usuario que se inicie sesión con la finalidad de poder asociar este certificado a una cuenta del servicio; para las peticiones posteriores, el certificado ya contará con una cuenta asociada a la cual podrá dar acceso siempre y cuando el certificado sea válido.

3.8.2. Estudio de requerimientos.

3.8.2.1. Requerimientos funcionales.

CIII_RF1. Primer inicio de sesión con chaffing. El servicio deberá guardar el certificado en caso de que sea la primera vez que le llega y si es válido, asociandolo a un usuario por medio de un inicio de sesión.

CIII_RF2. Inicio de sesión con chaffing. El servicio web, en caso de que el certificado sea válido, deberá de dar acceso a la cuenta del usuario sin la necesidad de tener que pedir las credenciales del mismo, a excepción de lo establecido en CIII_RF1. Primer inicio de sesión con chaffing.

CIII_RF3. Negación del inicio de sesión. En caso de que el certificado de usuario no sea válido, el servicio deberá negar el acceso al usuario.

CIII_RF4. Envío de chaffing y patrón de chaffing. El servicio web deberá mandar, en caso de que los reciba, el chaffing y el patrón de chaffing al API para que ésta pueda realizar el winnowing sobre el chaffing.

CIII_RF5. Comunicación con autoridad certificadora. El API desarrollada en este componente, deberá tener comunicación con el Componente II. Servidor Autentificador para verificar la validez del certificado.

CIII_RF6. Generación de llaves. El API deberá tener su par de llaves, pública y privada, para poder realizar el cifrado asimétrico al '*'patrón de chaffing'*'.

CIII_RF7. Descifrado del patrón de chaffing. El API deberá ser capaz de descifrar el patrón de chaffing con su llave privada, para poder

realizar el winnowing sobre el chaffing.

CIII_RF8. Realización de la etapa de winnowing. El API deberá de poder realizar la etapa de winnowing para poder obtener el certificado.

CIII_RF9. Validación de firma. Posteriormente a la etapa de winnowing del CIII_RF8, este tiene que validar que la firma del certificado, para saber si este fue emitido por nuestro mismo servidor autenticador.

CIII_RF10. Retorno de certificado. La API deberá de retornarle el certificado y su estatus al servicio para que este pueda realizar la lógica del negocio necesaria para el inicio de sesión.

CIII_RF11. Descifrado AES al patrón. La API deberá de descifrar el patrón utilizando AES.

3.8.2.2. Requerimientos no funcionales.

CIII_RNF1. Plataforma de implementación. Las pruebas del servidor web serán realizadas en el servidor Apache Tomcat.

CIII_RNF2. Base de datos. Se utilizará una base de datos en MySQL para guardar la información de los usuarios.

CIII_RNF3. Tamaño de llaves. Se utilizarán llaves de un tamaño de 2048 bits para el cifrado de tipo RSA.

CIII_RNF4. Framework de desarrollo. Se utilizará Spring Framework Core 5 para implementar la API con JAVA 8.

3.8.3. Reglas del negocio.

CIII_RN1. El servicio debe contar con una correcta conexión. Se debe de contar con una conexión estable tanto a internet para que los usuarios finales puedan acceder como al servidor autenticador necesario para el correcto funcionamiento.

CIII_RN2. Conexión a base de datos. El servidor debe tener una correcta conexión a la base de datos tanto del negocio como la modificación

necesaria para que el inicio de sesión implementado funcione correctamente.

CIII_RN3. API en el servidor. El servidor debe de contar con la API para que pueda realizar el procesos de *winnowing*.

CIII_RN4. Inicio de Sesión. El servidor debe contar con dos inicios de sesión, uno propio del negocio y otro propio del inicio de sesión con *Chaffing and Winnowing*.

Capítulo 4

Diseño.

4.1. Componente I: Extensión.

4.1.1. Diagrama de casos de uso

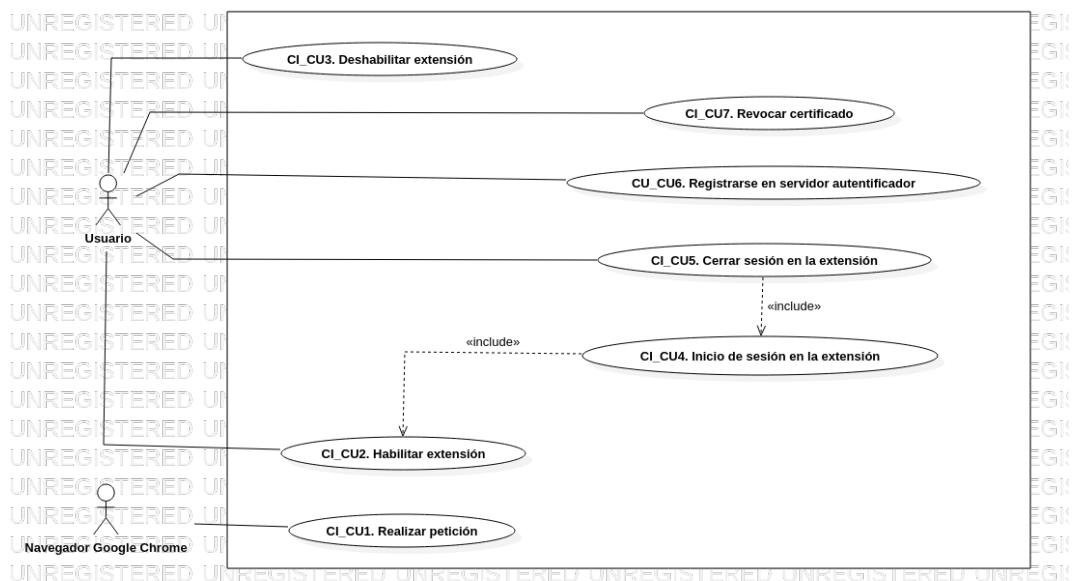


Figura 4.1: Diagrama de casos de uso del Componente I.

4.1.1.1. Descripción de casos de uso.

Caso de uso: CI_CU1. Realizar petición.	
Concepto	Descripción
Actor	Navegador de Google Chrome.
Propósito	Este caso de uso permite al navegador realizar una petición HTTP, ordenada por el usuario o un sistema externo.
Entradas	URL del servicio web solicitado.
Salidas	Petición HTTP.
Pre-condiciones	Algún agente externo (Sistema o usuario) ha ordenado al navegador mandar una petición HTTP.
Post-condiciones	Creación de la petición HTTP.
Reglas del negocio	-
Errores	La petición no se pudo realizar. La petición no es tipo HTTP.

Cuadro 4.1: Descripción CU: CI_CU1

... Trayectoria Principal ...

1. **El Usuario o El Sistema Externo** realiza una petición HTTP en el navegador Google Chrome.
2. **El navegador** realiza la petición.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. **El Usuario o El Sistema Externo** realiza una petición que no es HTTP en el navegador Google Chrome.
2. **El navegador** realiza la petición.

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CI_CU2. Habilitar extensión.	
Concepto	Descripción
Actor	Usuario.
Propósito	Este caso de uso, permite al usuario habilitar la extensión, para que ésta sea capaz de ver todas las peticiones que realiza el navegador.
Entradas	Indicación de habilitar extensión, mediante interfaz de usuario.
Salidas	-
Pre-condiciones	CI_CU3.
Post-condiciones	-
Reglas del negocio	CI_RN1.
Errores	No se puede habilitar la extensión.

Cuadro 4.2: Descripción CU: CI_CU2

... Trayectoria Principal ...

1. **El usuario** da click en el ícono de la extensión.
2. **El usuario** da click en el botón 'Activar'.
3. **La extensión** empieza a vigilar las peticiones que se realicen a través del navegador.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. **La extensión** no muestra el botón 'Activar', por ende el usuario no puede dar click.

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CI_CU3. Deshabilitar extensión.	
Concepto	Descripción
Actor	Usuario.
Propósito	Este caso de uso permite al usuario deshabilitar la extensión, para que ésta ignore todas las peticiones que se realicen por medio del navegador.
Entradas	Indicación de deshabilitar extensión, mediante interfaz de usuario.
Salidas	-
Pre-condiciones	CI_CU2.
Post-condiciones	-
Reglas del negocio	CI_RN2.
Errores	No se puede deshabilitar la extensión.

Cuadro 4.3: Descripción CU: CI_CU3

... Trayectoria Principal ...

1. **El usuario** da click en el ícono de la extensión.
2. **El usuario** da click en el botón.
3. **La extensión** deja de vigilar las peticiones que se realicen a través del navegador.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. **La extensión** no muestra el botón, por ende el usuario no puede dar click.

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CI_CU4. Inicio de sesión en la extensión	
Concepto	Descripción
Actor	Usuario
Propósito	<p>Este caso de uso permite al usuario poder iniciar sesión en la extensión para posteriormente obtener un código autentificador, es decir, el certificado del usuario, al cual se le aplicará <i>Chaffing</i>.</p> <p>Sólo se requerirá iniciar sesión una sola vez ya que después de ello el certificado se guardará en la extensión.</p>
Entradas	Usuario y Contraseña
Salidas	Certificado del usuario que acaba de iniciar sesión.
Pre-condiciones	Haber instalado la extensión en el navegador Google Chrome y haberla habilitado.
Post-condiciones	Por medio de los datos introducidos por el usuario, se obtendrá el certificado del mismo para que se pueda realizar la etapa de chaffing al momento que deseé identificarse en un servicio web.
Reglas del negocio	CI_RN1. CI_RN4. CI_RN7. CI_RN8. CI_RN9. CI_RN10. CI_RN11.
Errores	No se encuentra usuario registrado. No se puede obtener el código autentificador. No se pudo guardar el código autentificador. Contraseña no válida. Email no válido.

Cuadro 4.4: Descripción CU: CI_CU4

... Trayectoria Principal ...

1. ***El usuario*** da click en el ícono de la extensión.
2. ***EL usuario*** da click en el botón 'Iniciar sesión'.
3. ***La extensión*** abre en una nueva pestaña del navegador la página principal de la extensión.

4. ***El usuario*** llena correctamente los campos del formulario, los cuales son: 'Email' y 'Contraseña'.
5. ***EL usuario*** da click en el botón 'Iniciar sesión'.
6. ***La extensión*** despliega un popup con la información de la obtención del certificado.
7. ***El usuario*** da click en el botón 'Aceptar'.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. ***El usuario*** da click en el ícono de la extensión.
2. ***EL usuario*** da click en el botón 'Iniciar sesión'.
3. ***La extensión*** abre en una nueva pestaña del navegador la página principal de la extensión.
4. ***El usuario*** da click en el botón 'Iniciar sesión'.
5. ***La extensión*** abre la página de inicio de sesión.
6. ***El usuario*** llena correctamente los campos del formulario, los cuales son: 'Email' y 'Contraseña'.
7. ***EL usuario*** da click en el botón 'Iniciar sesión'.
8. ***La extensión*** despliega un popup con la información de la obtención del certificado.

... Fin de la Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. ***El usuario*** da click en el ícono de la extensión.
2. ***EL usuario*** da click en el botón 'Iniciar sesión'.
3. ***La extensión*** abre en una nueva pestaña del navegador la página principal de la extensión.
4. ***El usuario*** llena incorrectamente al menos un campo del formulario.

5. ***EL usuario*** da click en el botón 'Iniciar sesión'.
6. ***La extensión*** despliega un popup en donde informa cuál campo está llenado de forma incorrecta.
7. ***El usuario*** da click en el botón 'Aceptar'.
8. ***La extensión*** retorna al formulario sin realizar ninguna otra acción.

... Fin de la Trayectoria Alternativa 2 ...

... Trayectoria Alternativa 3 ...

1. ***El usuario*** da click en el ícono de la extensión.
2. ***EL usuario*** da click en el botón 'Iniciar sesión'.
3. ***La extensión*** abre en una nueva pestaña del navegador la página principal de la extensión.
4. ***El usuario*** da click en el botón 'Iniciar sesión'.
5. ***La extensión*** abre la página de inicio de sesión.
6. ***El usuario*** llena incorrectamente al menos un campo del formulario.
7. ***EL usuario*** da click en el botón 'Iniciar sesión'.
8. ***La extensión*** despliega un popup en donde informa cuál campo está llenado de forma incorrecta.
9. ***El usuario*** da click en el botón 'Aceptar'.
10. ***La extensión*** retorna al formulario sin realizar ninguna otra acción.

... Fin de la Trayectoria Alternativa 3 ...

Caso de uso: CI_CU5. Cerrar sesión en la extensión.	
Concepto	Descripción
Actor	Usuario
Propósito	Este caso de uso permite al usuario cerrar su sesión en el ordenador que se encuentre en ese momento, siempre y cuando haya iniciado sesión anteriormente. El objetivo es que se elimine su certificado de la extensión.
Entradas	-
Salidas	Sesión cerrada, en este momento la extensión se encuentra sin usuario logueado.
Pre-condiciones	CI_CU4.
Post-condiciones	La extensión debe mantenerse habilitada para su correcto funcionamiento.
Reglas del negocio	CI_RN1. CI_RN4. CI_RN5.
Errores	No se puede cerrar sesión.

Cuadro 4.5: Descripción CU: CI_CU5

... Trayectoria Principal ...

1. **El usuario** da click en el ícono de la extensión.
2. **El usuario** da click en el botón 'Cerrar sesión'.
3. **La extensión** cierra la sesión del usuario.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. **El usuario** da click en el ícono de la extensión.
2. **El usuario** da click en el botón 'Cerrar sesión'.
3. **La extensión** no puede cerrar la sesión y lo informa al usuario por medio de un popup.

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CI_CU6. Registrarse en servidor autentificador.	
Concepto	Descripción
Actor	Usuario
Propósito	Este caso de uso permitirá al usuario poder registrarse en el servidor autentificador si es que no tiene una cuenta con la cual poder iniciar sesión.
Entradas	email y contraseña ingresados por el usuario en la extensión.
Salidas	Estatus de operación.
Pre-condiciones	-
Post-condiciones	-
Reglas del negocio	CI_RN5. CI_RN7. CI_RN8. CI_RN9. CI_RN10. CI_RN11. CI_RN12. CI_RN13.
Errores	Error en el nombre de usuario. Error en el email. Error en la contraseña. No se pudo registrar al usuario.

Cuadro 4.6: Descripción CU: CI_CU6

... Trayectoria Principal ...

1. **El usuario** da click en el ícono de la extensión.
2. **La extensión** despliega un popup.
3. **El usuario** da click en el botón 'Iniciar Sesión'.
4. **La extensión** abre una página en otra pestaña del navegador.
5. **El usuario** da click en el botón 'Registrarse'.
6. **La extensión** muestra la página para registrarse.
7. **El usuario** llena los datos del formulario correctamente, los cuales son: 'Email', 'Contraseña' y 'Repetir contraseña'.

8. ***El usuario*** da click en el botón 'Crear Usuario'.
9. ***La extensión*** despliega un mensaje de confirmación.
10. ***El usuario*** da click en el botón '!Si, continuar!'.
11. ***La extensión*** despliega el estado de la operación.
12. ***El usuario*** da click en el botón 'Iniciar sesión'.
13. ***La extensión*** redirige a la página de inicio de sesión.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. ***El usuario*** da click en el ícono de la extensión.
2. ***La extensión*** despliega un popup.
3. ***El usuario*** da click en el botón 'Iniciar Sesión'.
4. ***La extensión*** abre una página en otra pestaña del navegador.
5. ***El usuario*** da click en el botón 'Registrarse'.
6. ***La extensión*** muestra la página para registrarse.
7. ***El usuario*** llena al menos un campo del formulario incorrectamente.
8. ***El usuario*** da click en el botón 'Crear Usuario'.
9. ***La extensión*** despliega un popup en donde informa cuál campo está llenado de forma incorrecta.
10. ***El usuario*** da click en el botón 'Aceptar'.
11. ***La extensión*** retorna al formulario sin realizar ninguna otra acción.

... Fin de la Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. **El usuario** da click en el ícono de la extensión.
2. **La extensión** despliega un popup.
3. **El usuario** da click en el botón 'Iniciar Sesión'.
4. **La extensión** abre una página en otra pestaña del navegador.
5. **El usuario** da click en el botón 'Registrarse'.
6. **La extensión** muestra la página para registrarse.
7. **El usuario** llena los datos del formulario correctamente, los cuales son: 'Nombre de usuario', 'Email', 'Contraseña' y 'Repetir contraseña'.
8. **El usuario** da click en el botón 'Crear Usuario'.
9. **La extensión** despliega un popup de confirmación.
10. **El usuario** da click en el botón 'Cancelar'.
11. **La extensión** retorna al formulario sin realizar ninguna otra acción.

... Fin de la Trayectoria Alternativa 2 ...

Caso de uso: CI_CU7. Revocar certificado.	
Concepto	Descripción
Actor	Usuario
Propósito	El usuario tendrá la opción de actualizar su certificado generando uno nuevo, con la finalidad de que el usuario tenga un mejor control de sus credenciales.
Entradas	Certificado autentificador.
Salidas	Certificado autentificador.
Pre-condiciones	CI_CU4, CI_CU6.
Post-condiciones	CI_CU4
Reglas del negocio	CLRN4. CLRN5. CLRN6.
Errores	No se puede almacenar el certificado autentificador.

Cuadro 4.7: Descripción CU: CI_CU7

... Trayectoria Principal ...

1. *La extensión* ha creado el certificado autentificador.
2. *La extensión* actualiza las credenciales de su nuevo certificado.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. *La extensión* no ha creado el certificado autentificador.
2. *La extensión* no guarda el certificado autentificador en storage. 4.18

... Fin de la Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. *La extensión* ha creado el certificado autentificador.
2. *La extensión* no puede guardar el certificado autentificador en storage.
3. *La extensión* muestra al usuario el siguiente mensaje "No se pudo guardar el certificado en el Storage" en la figura 4.18

... Fin de la Trayectoria Alternativa 2 ...

... Trayectoria Alternativa 3 ...

1. *La extensión* ha creado el certificado autentificador.
2. *La extensión* no puede eliminar el certificado anterior.
3. *La extensión* muestra al usuario el siguiente mensaje "No se pudo guardar el certificado en el Storage" en la figura 4.18

... Fin de la Trayectoria Alternativa 2 ...

4.1.2. Diagrama de flujo.

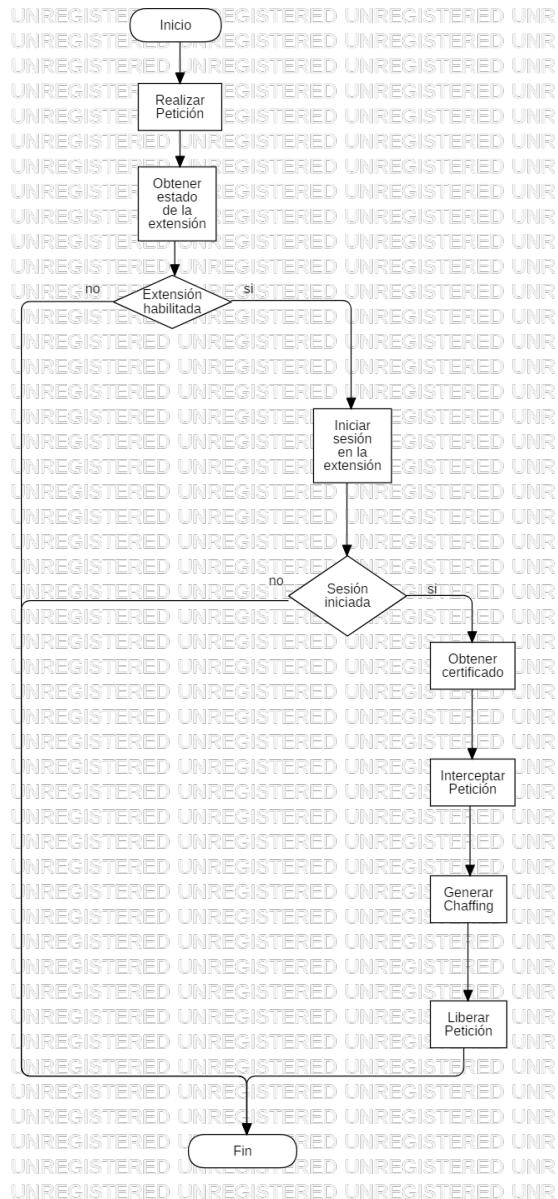


Figura 4.2: Diagrama de flujo del Componente I.

4.1.2.1. Descripción diagrama de flujo.

Para el caso de este diagrama se inicia con una petición realizada por el navegador web para luego analizar si la extensión se encuentra activada, en

caso de que no se realiza ninguna acción, pero si sí se encuentra, se analizará si ya se inició sesión, en caso de que no ya no se realiza ninguna acción pero en caso afirmativo se obtiene el certificado guardado en storage para luego interceptar el patrón y realizar el proceso de chaffing para finalmente liberar la petición modificada.

4.1.3. Diagrama de flujo de datos.

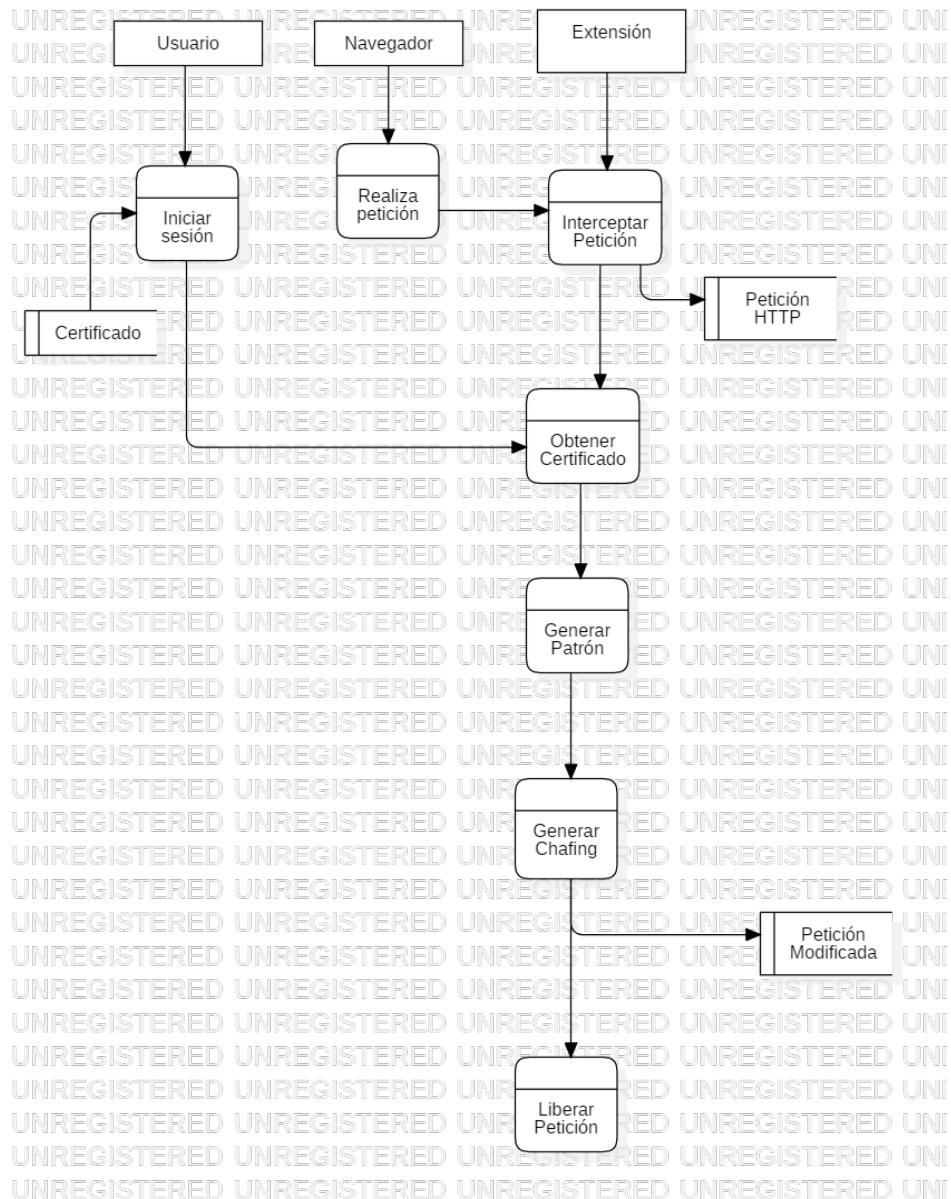


Figura 4.3: Diagrama de flujo de datos del Componente I.

4.1.3.1. Descripción diagrama de flujo de datos.

En este caso contamos con dos entidades externas, el usuario por una parte debe iniciar sesión para que se pueda generar un **Certificado** el cual

se utilizará para generar el chaffing y por otro lado el Navegador web que intercepta una **petición HTTP** que de igual manera se utilizará para generar el chaffing. Como resultado de la mezcla de estos dos se genera una **petición modificada** la cuál mas tarde se liberará para que pueda viajar hacia el servidor.

4.1.4. Diagrama de clases.

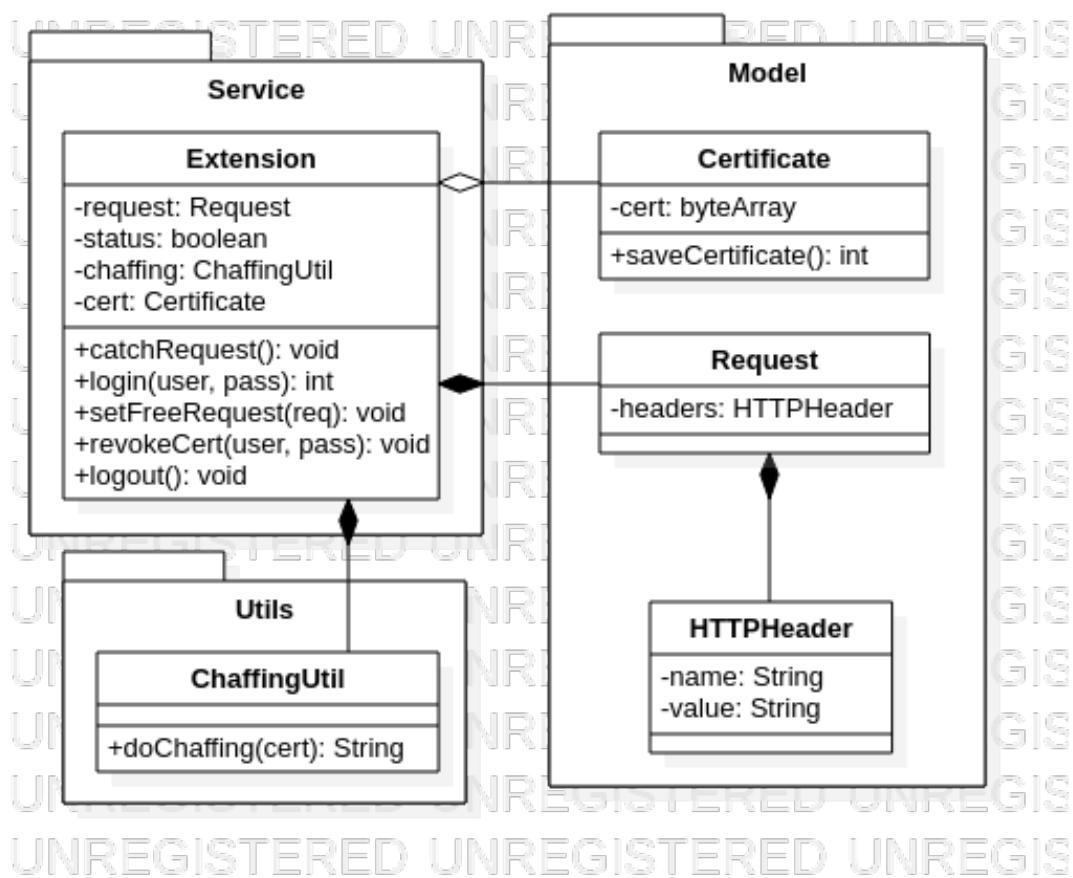


Figura 4.4: Diagrama de clases de Componente I.

4.1.4.1. Descripción de diagrama de clases

Clase: Extension

Atributos

1. **request** : Variable que almacena la petición HTTP.
 - Tipo de dato: **Request**.
2. **status** : Variable para saber si la extensión está activada o no.
 - Tipo de dato: **boolean**.
3. **chaffing** : instancia para ejecutar el proceso de Chaffing.
 - Tipo de dato: **ChaffingUtil**.
4. **cert** : Variable para almacenar el certificado del usuario.
 - Tipo de dato: **Certificate**.

Métodos

1. **catchRequest()**: Este método permite interceptar las peticiones del usuario.
 - Tipo de dato de retorno: **void**
2. **login(String user, String pass)**: Este método permite iniciar sesión para obtener el certificado del usuario.
 - Tipo de dato de retorno: **int**
3. **logout(String pass, String email)**: Este método permite cerrar sesión en la extensión y eliminar el certificado almacenado.
 - Tipo de dato de retorno: **void**
4. **revokeCert(String user, String pass)**: Este método permite revocar el certificado del usuario.
 - Tipo de dato de retorno: **void**
5. **setFreeRequest(Request req)**: Este método permite liberar la petición nueva con chaffing.
 - Tipo de dato de retorno: **void**

Clase: *ChaffingUtil*

Métodos

1. **doChaffing(Certificate certificate)**: Este método permite realizar la etapa de chaffing.

- Tipo de dato de retorno: **String**

Clase: *HTTPHeader*

Atributos

1. **name**: variable para guardar el nombre del header el protocolo.
 - Tipo de dato de retorno: **String**
2. **value**: variable para guardar el contenido del header el protocolo.
 - Tipo de dato de retorno: **String**

Clase: *HTTPRequest*

Atributos

1. **headers**: variable para guardar los header de una petición HTTP.
 - Tipo de dato de retorno: **HTTPHeader**

Clase: *Certificate*

Atributos

1. **cert**: variable para guardar el certificado del usuario.
 - Tipo de dato de retorno: **ByteArray**

Métodos

1. **saveCertificate()**: variable para guardar el certificado del usuario en el Storage.
 - Tipo de dato de retorno: **ByteArray**

4.1.5. Diagramas de secuencia.

4.1.5.1. Diagrama de secuencia: Realizar petición.

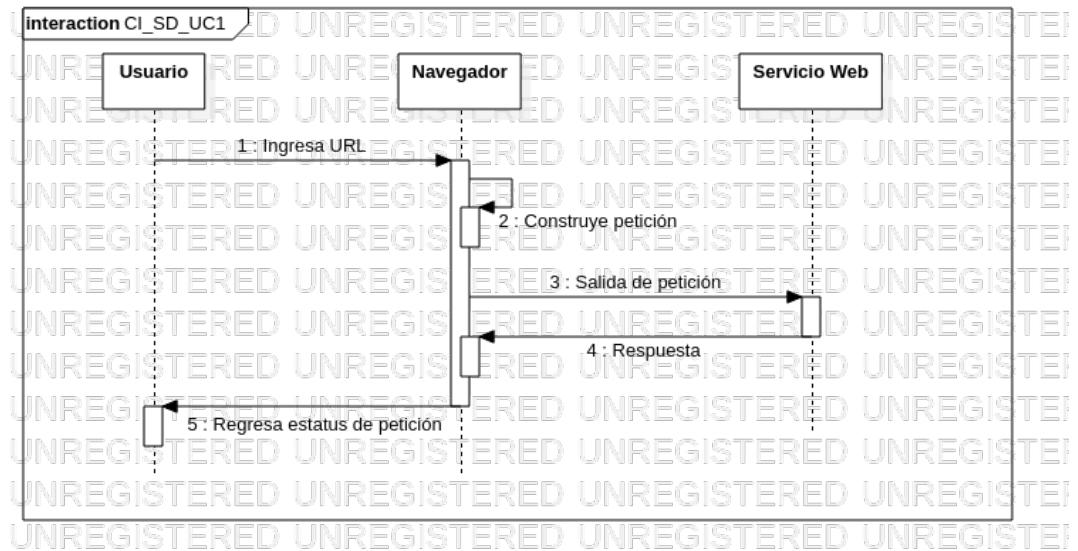


Figura 4.5: Diagrama de secuencia del CI_CU1. Realizar petición.

Descripción: Para realizar una petición, es necesario seguir la siguiente secuencia. Primeramente, el usuario ingresa un URL al navegador, para que este construya la petición que mandará. Una vez que ha construido la petición realiza la petición, es decir sale a red la petición.

4.1.5.2. Diagrama de secuencia: Habilitar extensión.

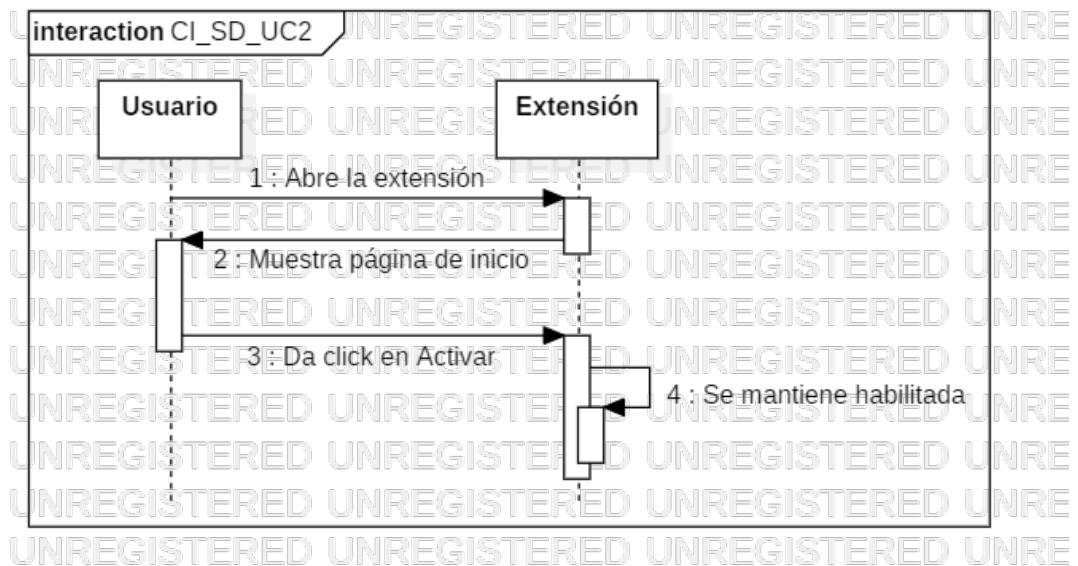


Figura 4.6: Diagrama de secuencia del CI_CU2. Habilitar extensión.

Descripción: Para habilitar la extensión, el usuario necesita primero abrir la extensión. Una vez abierta, debe de dar click en el botón 'Activar' para que la extensión ejecute la orden y se mantenga habilitada.

4.1.5.3. Diagrama de secuencia: Deshabilitar extensión.

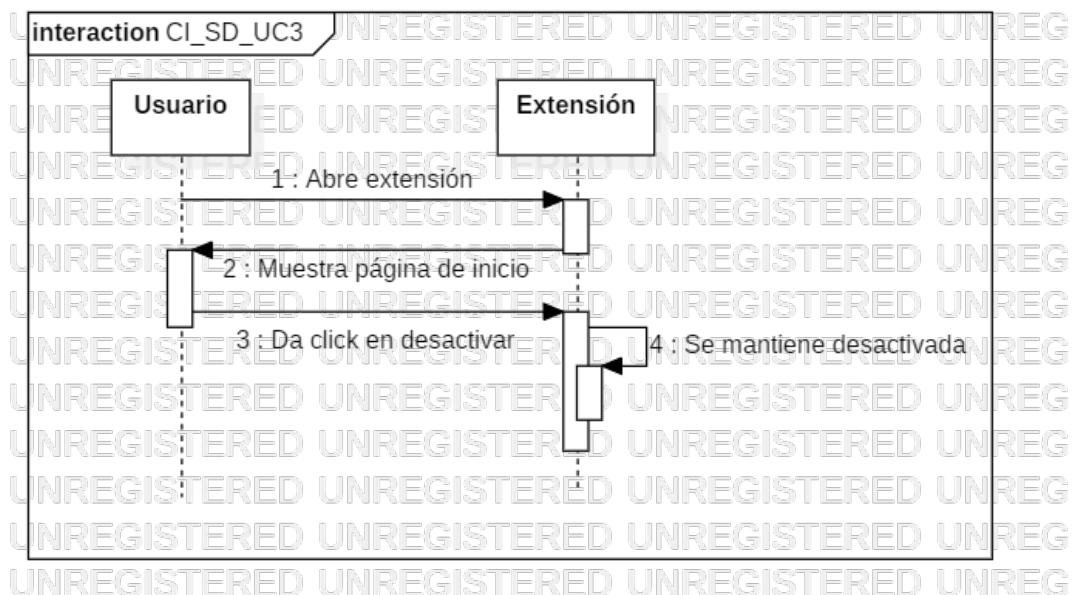


Figura 4.7: Diagrama de secuencia del CI_CU3. Deshabilitar extensión.

Descripción: Para deshabilitar la extensión, el usuario necesita primero abrir la extensión. Una vez abierta la extensión, debe de dar click en el botón 'Desactivar', para que la extensión ejecute la orden y se mantenga deshabilitada.

4.1.5.4. Diagrama de secuencia: Inicio de sesión

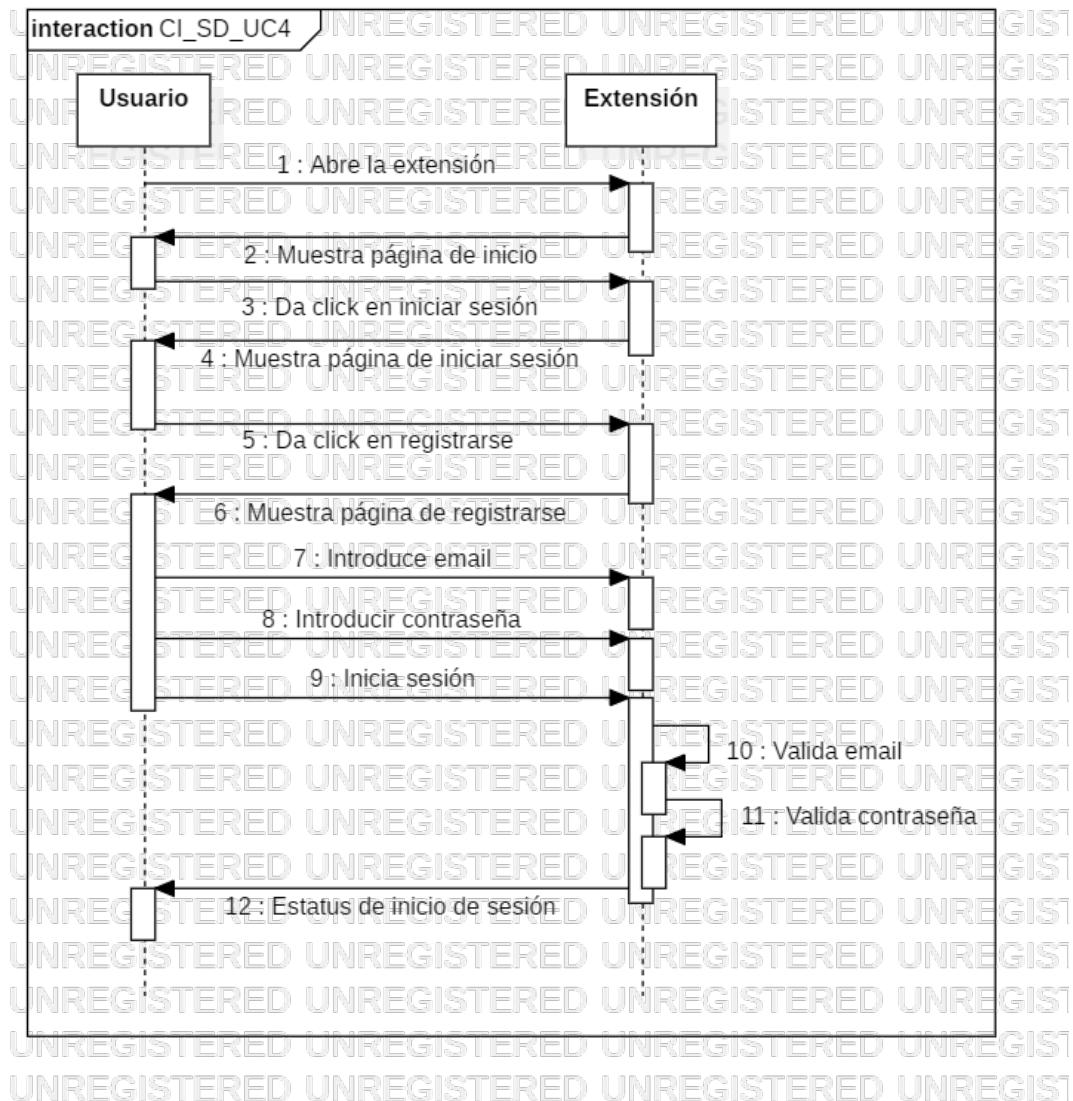


Figura 4.8: Diagrama de secuencia del CI_CU4. Inicio de sesión en la extensión.

Descripción: Como primeros dos pasos de la secuencia para iniciar sesión en la extensión, el usuario ingresa su nombre de usuario y su contraseña, para después iniciar sesión en la extensión. Una vez que el usuario ha dado la orden de iniciar sesión, la extensión valida el nombre de usuario y la contraseña, retornando un mensaje en caso de que alguno de estos dos valores esté mal,

si no es el caso, la extensión retornará un mensaje con el estatus del inicio de sesión.

4.1.5.5. Diagrama de secuencia: Cerrar sesión.

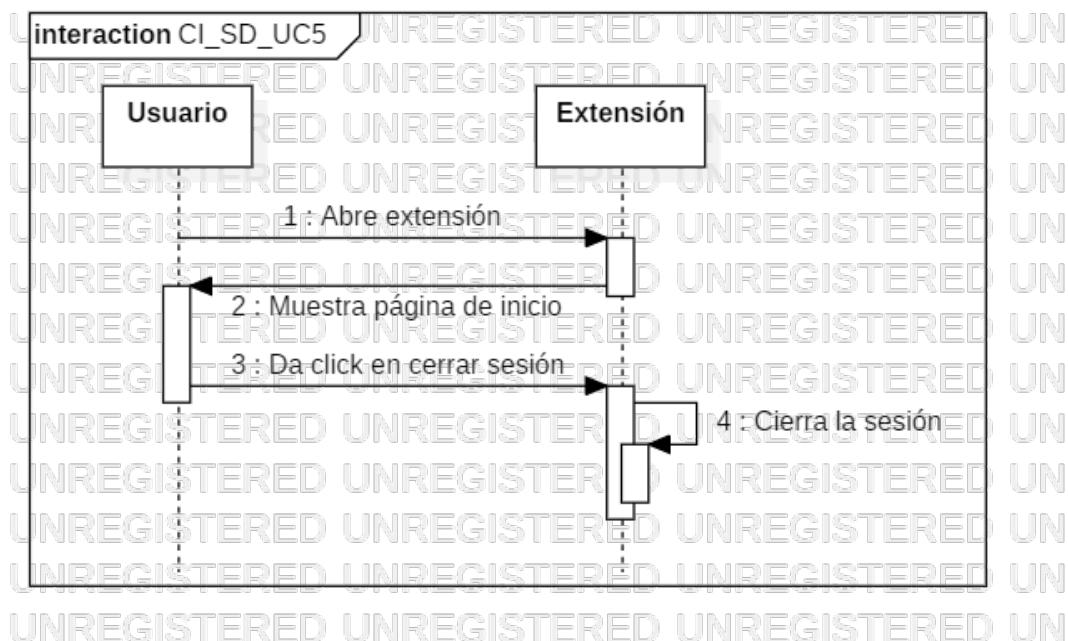


Figura 4.9: Diagrama de secuencia del CI_CU5. Cerrar sesión en la extensión.

Descripción: Como primer paso de este diagrama de secuencia se tiene que abrir la extensión para después dar click en el botón de 'Cerrar Sesión'.

4.1.5.6. Diagrama de secuencia: Registrarse en servidor autenticador.

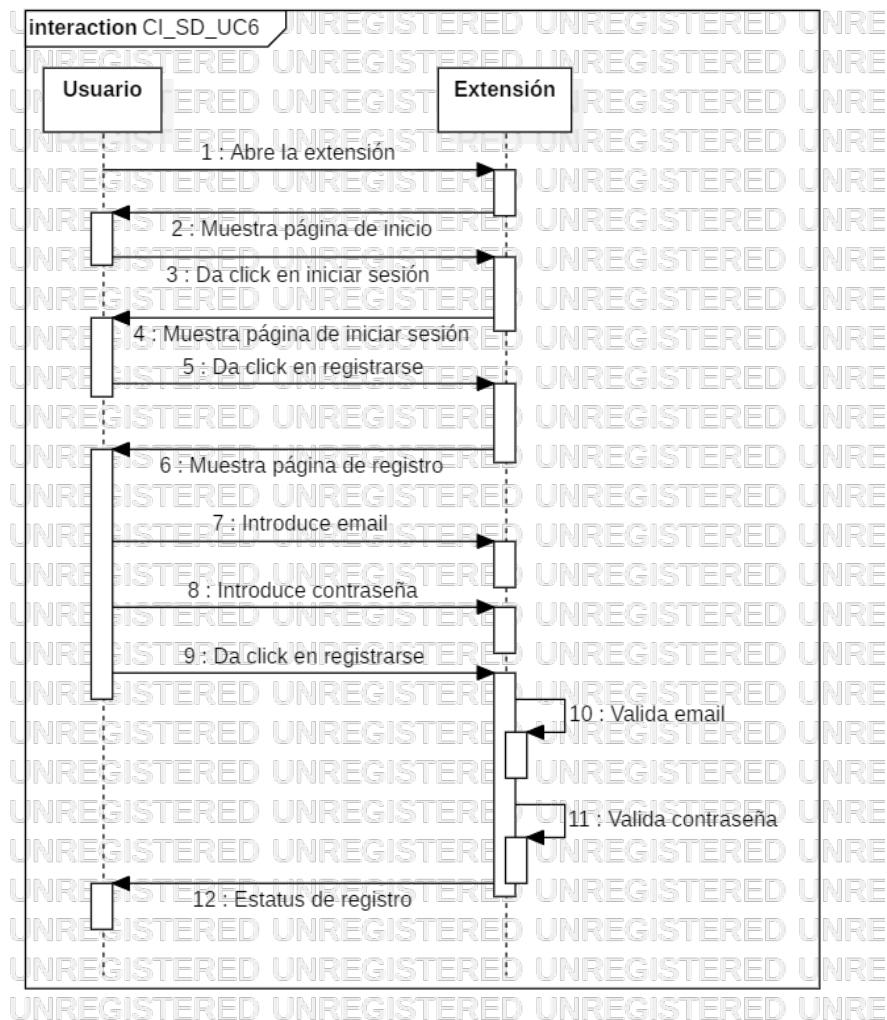


Figura 4.10: Diagrama de secuencia del CI_CU6. Registrarse en servidor autenticador.

Descripción: Como primeros pasos de este diagrama de secuencia, la extensión se tiene que abrir para después dar click en el botón de 'Iniciar sesión' para desplegar la página principal y, una vez ahí, dar click en el botón de 'Registrarse'. Una vez en la página de registro, el usuario, como siguiente paso, llena los datos requeridos por el formulario para después dar click en el

botón 'Crear Usuario'. La extensión validará los campos retornando un mensaje de error en caso de que alguno se llene mal, si no es el caso, retornará un aviso con el estatus del registro.

4.1.5.7. Diagrama de secuencia: Revocar certificado.

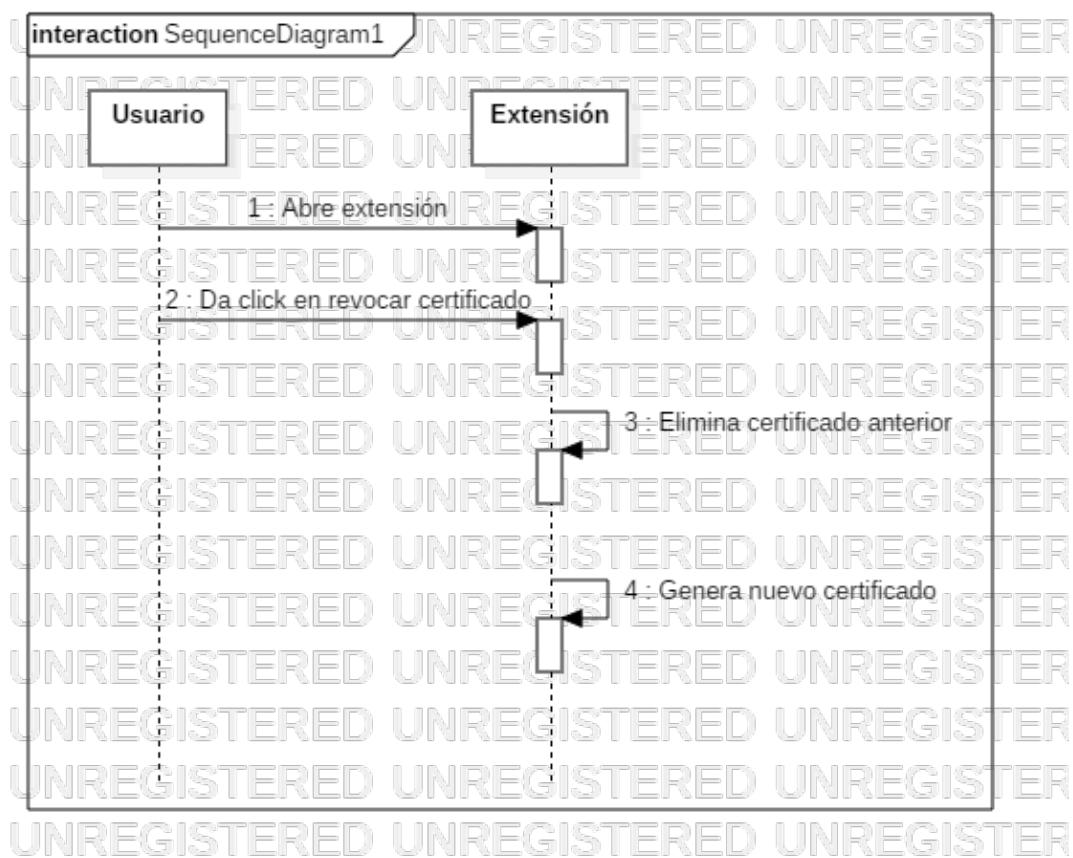


Figura 4.11: Diagrama de secuencia del CI_CU7. Revocar certificado.

Descripción:

4.1.6. Diagrama de actividades

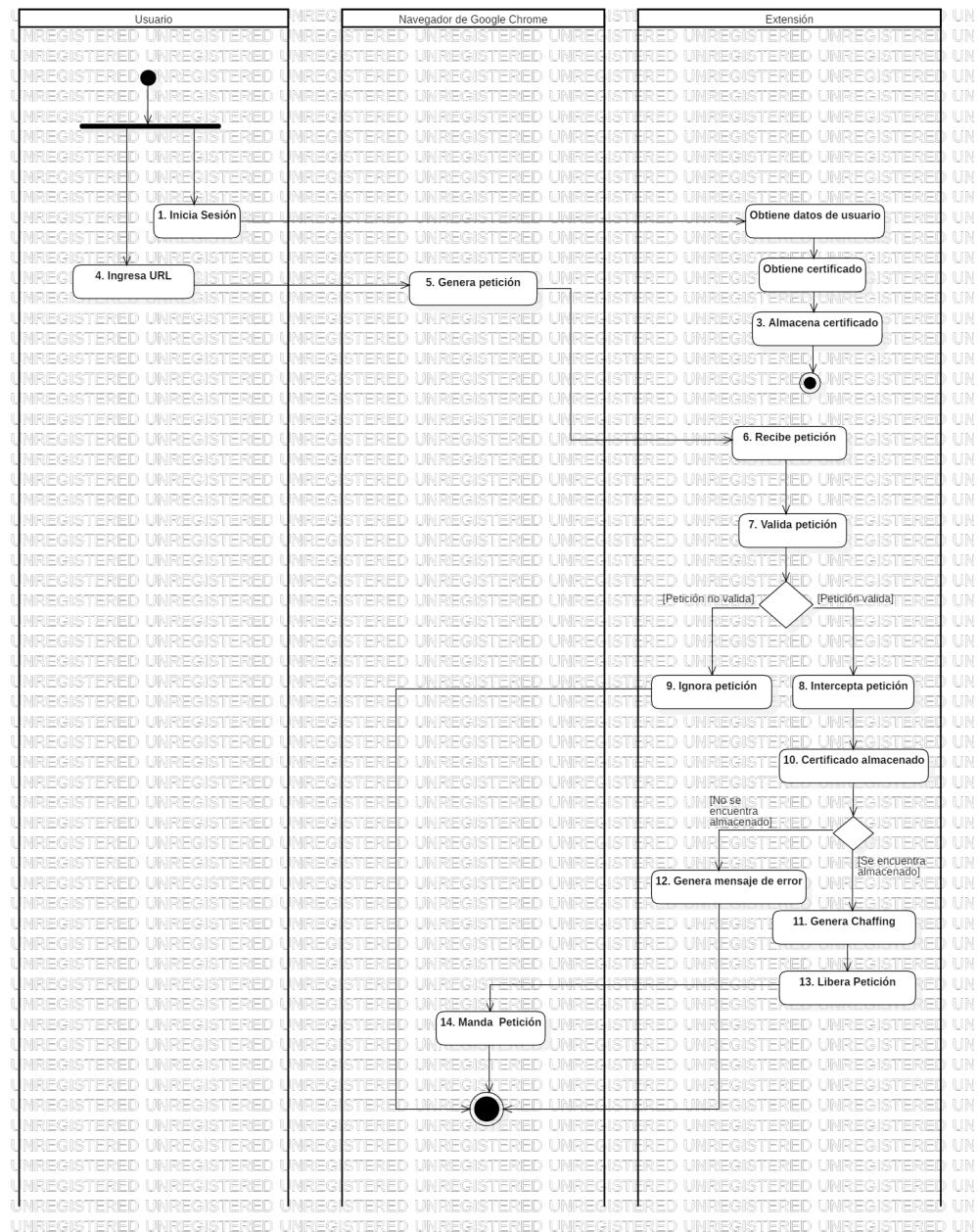


Figura 4.12: Diagrama de actividades del Prototipo 2.

4.1.6.1. Descripción del diagrama de actividades.

El componente cuenta con los siguientes pasos. Uno de los primeros pasos que debe hacer el usuario es iniciar sesión, esto con el fin de generar un certificado de acuerdo a los datos del usuario. Dicho certificado se almacena en el **Storage** de Google Chrome.

El otro camino a seguir por el usuario es realizar una búsqueda en el navegador, en la cual el navegador realiza la petición y la extensión recibe la misma. Modificándola siempre y cuando se encuentre un certificado guardado en el Storage. Si este proceso es válido, se genera el proceso de Chaffing y se libera la petición.

4.1.7. Interfaz de usuario.

4.1.7.1. Pantalla Inicial.

Esta pantalla es la primer vista que el usuario tiene el sistema. Aparece al darle click a la extensión en su ícono correspondiente.



Figura 4.13: Pantalla de interfaz de la extensión.

En ella, el usuario puede activar y desactivar la extensión para empezar a interceptar peticiones, ademas cuenta con un botón de inicio de sesión (*Iniciar Sesión*), el cual al darle click abrirá la pantalla 'Tab de la extensión' (Figura 4.14). La cual se muestra a continuación.

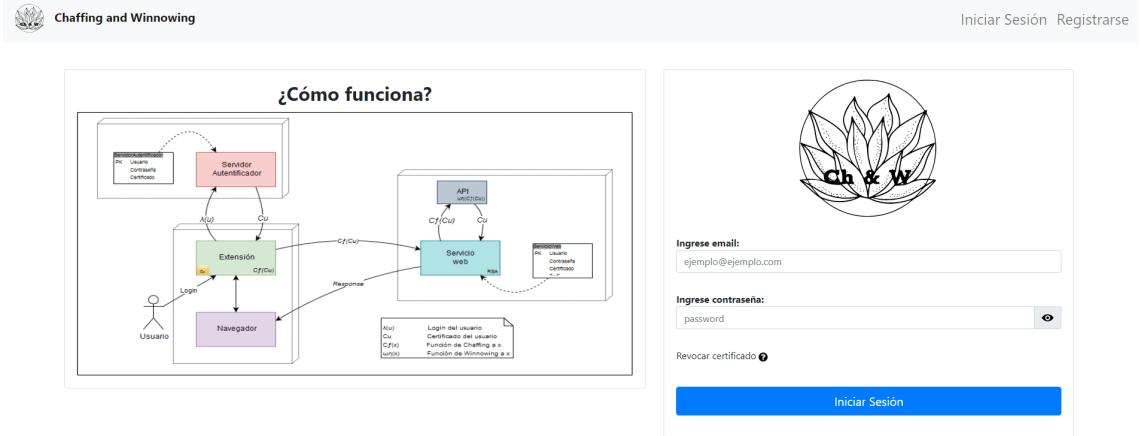


Figura 4.14: Pantalla inicial.

El navegador Google Chrome tiene ciertas restricciones con los certificados que no son firmados por una autoridad certificadora de confianza, y debido a que nuestros certificados son autofirmados. nos aparecerá un mensaje de alerta por parte del navegador, para evitar que esto sea un problema, en la extensión vamos a seleccionar la opción de *Aviso*, donde nos abrirá una pestaña nueva de un aviso de seguridad, vamos a escoger la opción de *opciones avanzadas*, y se desplegará información adicional con la opción de *Proceder a la ip x.x.x.x*, como se muestra en la siguiente imagen:



Your connection is not private

Attackers might be trying to steal your information from **172.16.140.79** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

Help improve Safe Browsing by sending some [system information](#) and [page content](#) to Google.
[Privacy policy](#)

[Hide advanced](#)

[Back to safety](#)

This server could not prove that it is **172.16.140.79**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to 172.16.140.79 \(unsafe\)](#)

Figura 4.15: Pantalla de aviso.

4.1.7.2. Tab de la extensión.

Esta pantalla es la interfaz que se le brinda al usuario para que inicie sesión y obtenga su certificado autenticador (Figura 4.16). En ella se aprecian únicamente dos campos para introducir texto ('Ingrrese email' e 'Ingrrese contraseña'), y un botón 'Iniciar Sesión'.

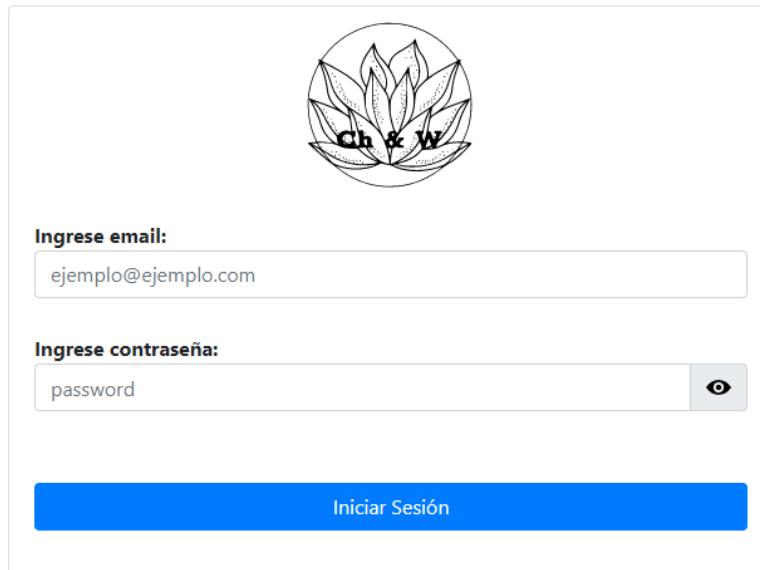


Figura 4.16: Tab de la extensión. Permite inicio de sesión

Durante el inicio de sesión el usuario puede visualizar mensajes de éxito o error. La Figura 4.17 se le muestra al usuario tras haber obtenido el certificado desde la autoridad y guardado con éxito en el Storage de Google Chrome. La Figura 4.18 notifica al usuario que existió un error al obtener el certificado de la autoridad certificadora.

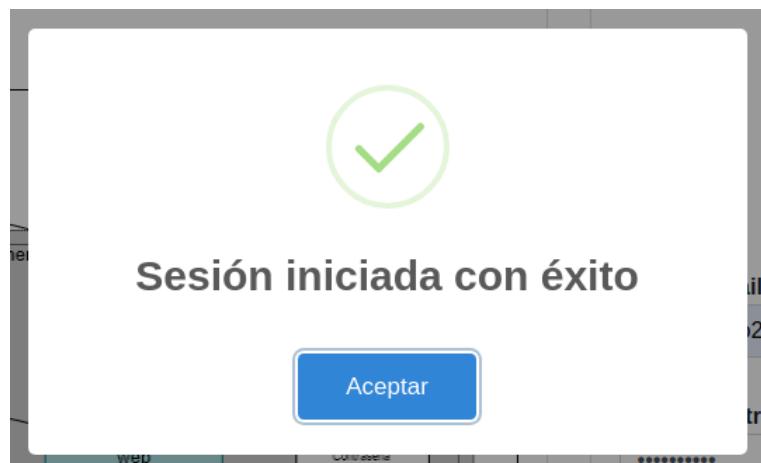


Figura 4.17: Mensaje de éxito tras obtener el certificado de la autoridad y guardarlo en el storage de Google Chrome.

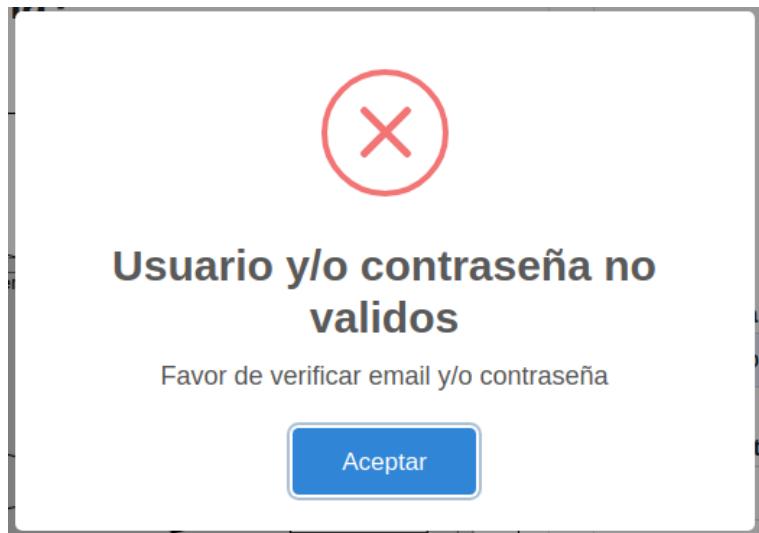


Figura 4.18: Mensaje de error al tratar de obtener certificado de la autoridad (Usuario y/o contraseña incorrectos).

Por otra parte, se le brinda al usuario una interfaz donde pueda registrarse, para ello basta con escoger la opción de *Registrarse* en la parte superior de la interfaz, al acceder a esta interfaz, aparecerán cuatro campos los cuales el usuario puede llenar para crear una cuenta y registrarse en la extensión. Estos campos son correo electrónico y contraseña.



Ingrese email:
ejemplo@ejemplo.com

Ingrese contraseña: ⓘ
password

Repetir contraseña:
password

Crear Usuario

Figura 4.19: Tab de la extensión. Permite registrarse

Al llenar los campos y enviar la petición a la autoridad certificadora de registrarse, se le notificará al usuario si pudo generarse el registro del usuario con éxito u ocurrió un error tras crear el registro. Cabe resaltar que la autoridad creará un certificado para dicho usuario, el cual el usuario deberá obtener posteriormente con un inicio de sesión, éste proceso se explicará en el componente 2.

Como mensaje de éxito se muestra la figura 4.20 la cual significa que el usuario fue creado con éxito y se le generó un certificado para dicho usuario.

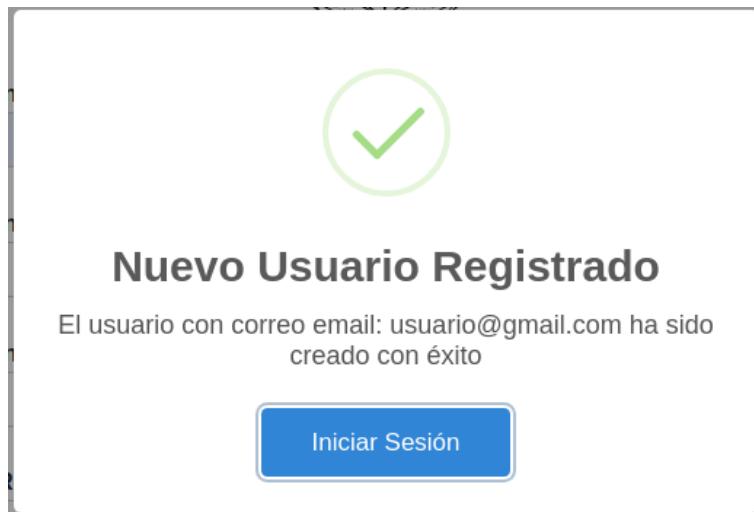


Figura 4.20: Mensaje de éxito tras obtener el certificado de la autoridad y guardarlo en el storage de Google Chrome.

Como mensajes de errores se muestran distintos tipos de mensajes para diferentes situaciones, las cuales se explican a continuación.

La figura 4.21 notifica al usuario que el email introducido no es valido, es decir, no cuenta con una estructura valida de un correo electrónico (email).

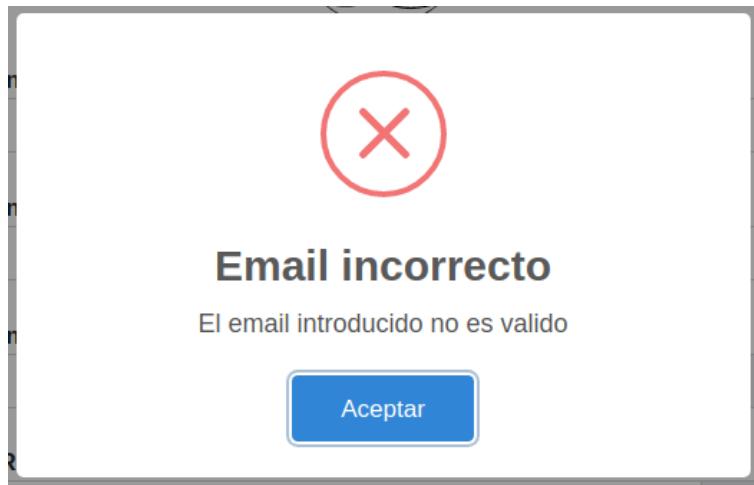


Figura 4.21: Mensaje de error tras detectar que no se cumplen los requisitos para el correo ingresado.

La figura 4.22 muestra un error tras no cumplir los requisitos con la

contraseña introducida.

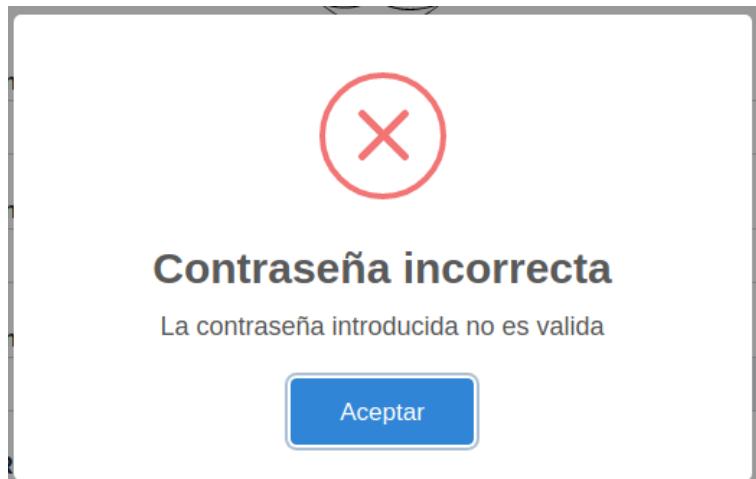


Figura 4.22: Mensaje de error tras detectar que no se cumplen los requisitos para la contraseña ingresada.

La figura 4.23 muestra un error tras no coincidir las contraseñas introducidas.

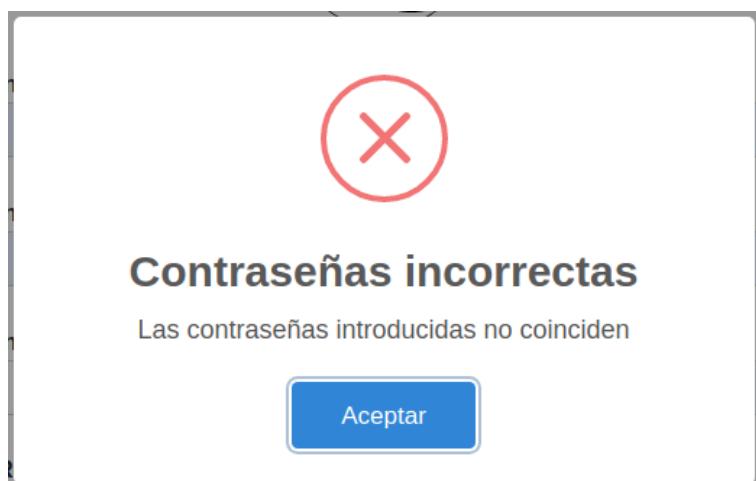


Figura 4.23: Mensaje de error tras detectar que no coinciden el par de contraseñas ingresadas.

Y por ultimo se muestra en la figura 4.24 el error que se obtiene al tratar de registrarse con un usuario que ya tiene una cuenta en la extensión.

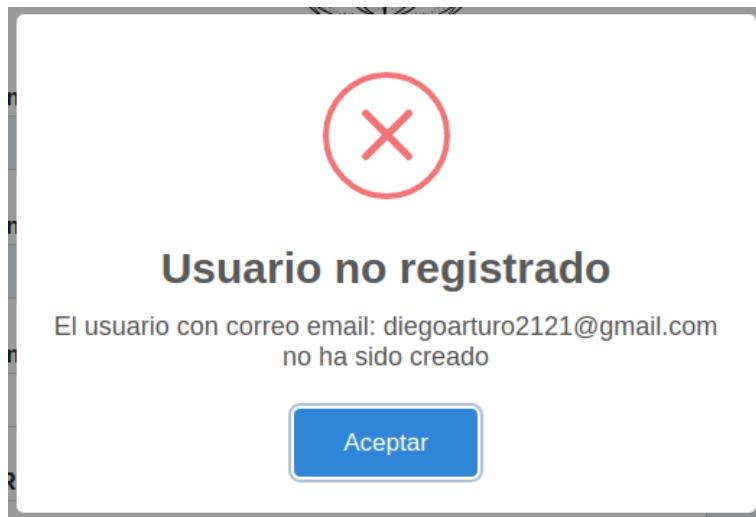


Figura 4.24: Mensaje de error tras detectar que el usuario ya se encuentra registrado.

El usuario tendrá también la posibilidad de revocar su certificado, permitiéndole así generar uno por la autoridad certificadora. La figura 4.25 muestra la interfaz que se le proporciona al usuario para realizar la revocación de su certificado.

A screenshot of a login form for certificate revocation. At the top center is a circular logo featuring a stylized plant or flower with the letters "Ch & W" in the center. Below the logo is a text field labeled "Ingrese email:" containing the placeholder "ejemplo@ejemplo.com". Below that is a password field labeled "Ingrese contraseña:" containing the placeholder "password" and a small eye icon to toggle visibility. At the bottom is a large blue rectangular button with the text "Revocar Certificado" in white.

Figura 4.25: Interfaz para revocar certificado.

La figura 4.26 muestra un mensaje de éxito, el cual se le da a conocer al usuario que su certificado fue revocado exitosamente.

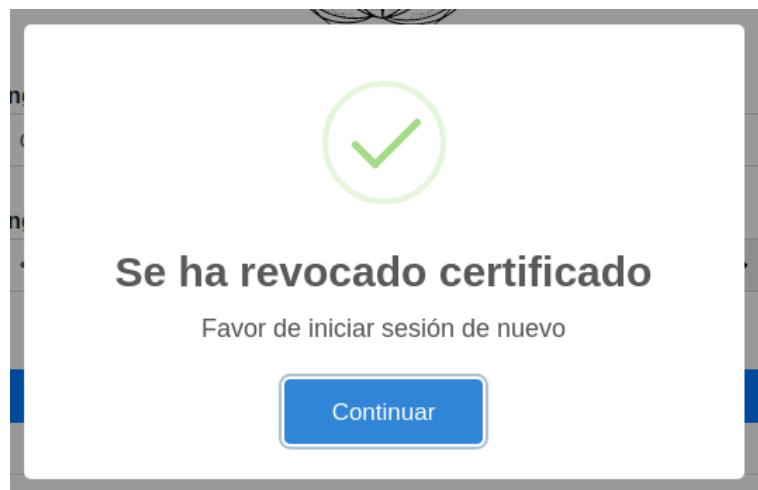


Figura 4.26: Mensaje de éxito al revocar el certificado de un usuario.

Si el usuario ingresa sus credenciales (correo electrónico y contraseña) incorrectas, la interfaz despliega la información de error de la figura 4.27.

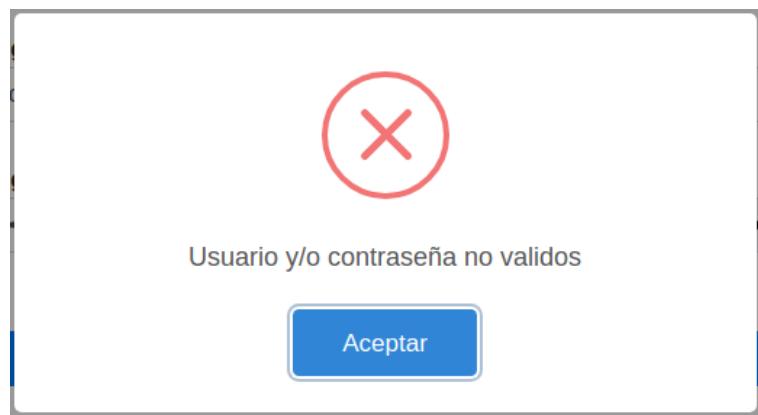


Figura 4.27: Mensaje de error al tratar de revocar el certificado de un usuario con credenciales incorrectas.

El usuario visualizará y tendrá interacción con este componente para obtener el certificado y llevar a cabo el proceso de Chaffing. Este proceso se lleva a cabo en segundo plano de la extensión (background) el cual cacha la

petición e inyecta el resultado del Chaffing en el header de la petición hecha previamente. Una vez injectado el código, la extensión libera la petición y ésta viaja en red al servidor de prueba.

4.1.8. Requisitos de diseño.

En este apartado, se especificarán los requisitos de diseño para que el prototipo opere de forma correcta, de igualmanera se tiene por entendido que son necesarios los Requisitos del diseño del primer prototipo.

4.1.8.1. Requisitos de ejecución de la extensión

Para poder ejecutar el prototipo dos es necesario contar con todo lo requerido para el prototipo uno y agregarle la interacción con **jQuery 3.3.1** y **Bootstrap** que son dos frameworks que corren sobre JavaScript y que nos permiten interactuar un poco mejor con el usuario haciendo la interfaz de la extensión más amigable y entendible, estos ya se encuentran insertados en la extensión por lo que no es necesario que el usuario realice acción alguna. Otros de los requisitos para su ejecución es contar con un **usuario y contraseña** válidos ya que serán de suma importancia para el correcto funcionamiento del prototipo dos. Para deshabilitar la extensión, el usuario necesita primero abrir la extensión. Una vez abierta la extensión, debe de dar click en el botón "deshabilitar", para que la extensión ejecute la orden y se mantenga deshabilitada.

4.1.8.2. Requisitos para el envío del código autentificador

Para este prototipo se considera necesaria los siguientes requisitos de diseño:

- **Código Autentificador** Archivo indispensable en este prototipo debido a que sin este es imposible completar el proceso.
- **Iniciar Sesión** Este botón generará un certificado de prueba y se almacenará en el Storage del navegador Google Chrome. Esto con el fin, que al momento de interceptar la petición, la extensión de encargará de inyectar el certificado almacenado modificado con el método *Chaffing* en el encabezado de protocolo HTTP.

4.1.9. Algoritmos.

Los algoritmos necesarios para hacer el *chaffing* son expuestos a continuación:

```
Data: lenCert, lenAccept, Cert  
Result: patternChaffing  
1 lenPattern  $\leftarrow$  lenCert + lenAccept;  
2 Pattern[lenPattern * 8]  $\leftarrow$  0;  
3 unosCert  $\leftarrow$  getOnes(Cert);  
4 for i = 1 to unosCert do  
5   repeat  
6     | random  $\leftarrow$  secure_random(0, lenPattern);  
7     | until Pattern[random]! = 0;  
8     | Pattern[random]  $\leftarrow$  '1';  
9 end
```

Algoritmo 1: getPattern: Generación de patrón de chaffing

```

Data: Cert, Accept
Result: Chaffing
1 lenCert  $\leftarrow$  length(Cert);
2 lenAccept  $\leftarrow$  length(Accept);
3 Pattern  $\leftarrow$  getPattern(lenCert, lenAccept, Cert);
4 Chaffing  $\leftarrow$  "";
5 countCert  $\leftarrow$  0;
6 countAccept  $\leftarrow$  0;
7 certBits  $\leftarrow$  getBits(Cert);
8 acceptBits  $\leftarrow$  getBits(Accept);
9 foreach i in Pattern do
10   if i ==' 1' then
11     Chaffing  $\leftarrow$  Chaffing + certBits[countCert];
12     countCert  $\leftarrow$  countCert + 1;
13   else
14     Chaffing  $\leftarrow$  Chaffing + acceptBits[countAccept];
15     countAccept  $\leftarrow$  countAccept + 1;
16   end
17 end
18 Chaffing  $\leftarrow$  getBytes(Chaffing);

```

Algoritmo 2: getChaff: Generación de chaffing

Primero vamos a analizar el algoritmo del *patrón de chaffing* con un pequeño ejemplo (utilizaremos datos de variables cortos); supongamos que nuestro certificado es el siguiente:

$$C_k = MITZ057abZ251$$

y utilizaremos el campo *Accept* del header de la petición como *chaff*, lo cual tendrá lo siguiente:

$$P_{HTTP} = Mozilla5,5/Chrome8,1/Safari$$

Entonces, nuestro patrón de chaffing terminará teniendo un tamaño de la longitud de los caracteres de nuestro certificado más la longitud de los datos de la petición HTTP, esta variable se llamará *lenPattern*. A continuación vamos a utilizar un arreglo de banderas de ese tamaño multiplicado por 8, para que así podamos llenar de ceros y unos este arreglo y cada una de las posiciones nos represente en el chaffing un bit, este arreglo inicia con 0 todos sus valores, y que al final será nuestro patrón, esta variable será *Pattern*.

Este algoritmo nos permite generar posiciones aleatorias dentro del rango del tamaño de *lenPattern*, y tantas veces como se tengan caracteres en el certificado, se pondrá un 1 en dicha posición si es que hay un 0, en caso de que no se encuentre un 0 se repetirá el procedimiento obteniendo una posición aleatoria diferente, esto nos generará nuestro patrón para el Chaffing. Utilizaremos un contador para conseguir esto, aumentándolo cada vez que se obtenga un número aleatorio válido. En nuestro algoritmo, nuestra variable *random* contiene la posición aleatoria donde se validará si en esa posición se puede poner un 1 o no.

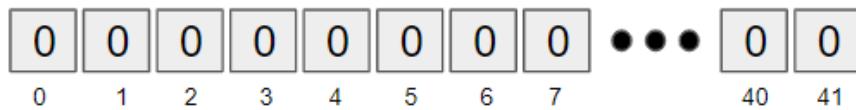


Figura 4.28: Arreglo del patrón de chaffing

Supongamos, por ejemplo, que el primer número aleatorio que se obtiene es 4, entonces en la posición 4 colocaremos un 1:



Figura 4.29: Arreglo del patrón de chaffing después de una iteración

Este algoritmo nos permite llenar de unos el mismo número de veces que el número de unos contenga nuestro certificado en bits, lo que nos asegura que cada posición le corresponde a un byte ya sea del certificado o de la petición a enviar. Supongamos que al terminar este algoritmo, nuestro arreglo queda algo parecido a lo siguiente:



Figura 4.30: Arreglo del patrón de chaffing final

El segundo algoritmo realizará la etapa de Chaffing, utilizando el patrón generado en el algoritmo anterior, recorriendo cada posición de este arreglo, contaremos con dos contadores para saber que carácter se debe de poner a continuación en el proceso del algoritmo, *countCert* y *countAccept*, contador para el certificado y para el encabezado Accept respectivamente.

Cuando se detecte un 1 en el arreglo del patrón, vamos a proceder a colcoar el byte siguiente correspondiente al arreglo del certificado. En caso análogo, si se encuentra con un 0, entonces se colocará el siguiente carácter del encabezado Accept, se llevará el control de ambos mediante sus respectivos contadores.

Por último, es importante mencionar que se mandará el Chaffing en base64 para evitar problemas con los caracteres no válidos que se puedan generar. Seguido del Chaffing en base64, vamos a enviar el patrón con un cifrado híbrido, cifrando primero el patrón con AES, la llave que utilizaremos para realizar este cifrado la cifraremos con RSA utilizando la llave pública del servidor.

4.1.9.1. Complejidad computacional.

Haciendo un análisis de los algoritmos anteriormente mostrados, deducimos que la complejidad del algoritmo de *chaffing* es: $O(n)$ Donde n es la longitud de caracteres del certificado.

4.2. Componente II: Servidor Autentificador.

4.2.1. Diagrama de casos de uso.

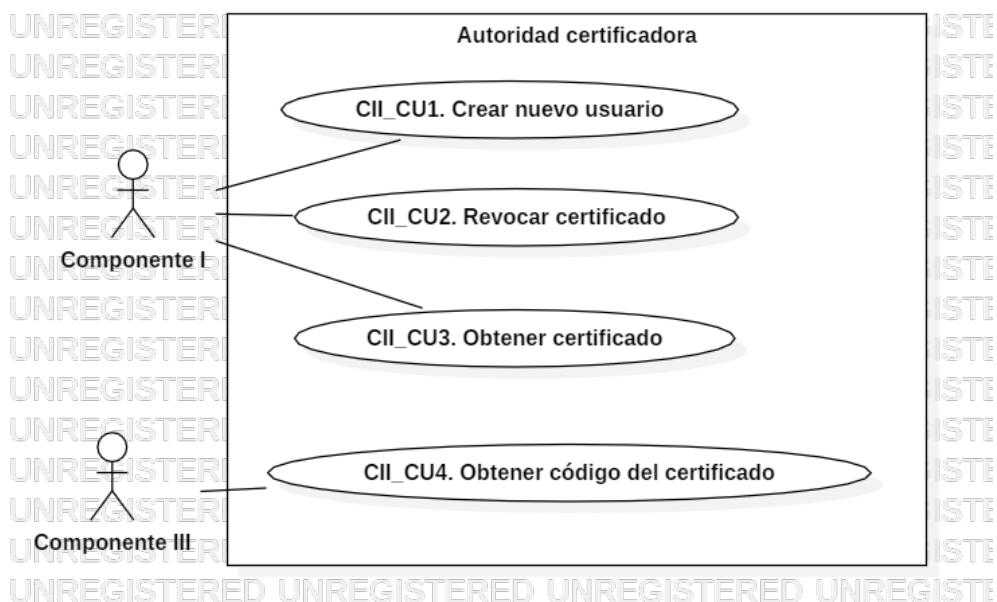


Figura 4.31: Diagrama de casos de uso del Componente II.

4.2.1.1. Descripción de casos de uso.

Caso de uso: CII_CU1. Crear nuevo usuario.	
Concepto	Descripción
Actor	Componente I. Extensión.
Propósito	Guardar un nuevo usuario en el Componente I para que éste pueda hacer uso de este método de autenticación.
Entradas	Correo electrónico y contraseña.
Salidas	Status de operación.
Pre-condiciones	-
Post-condiciones	Creación de un certificado.
Reglas del negocio	CII_RN1 CII_RN2
Errores	No se pudo registrar el usuario en la base de datos.

Cuadro 4.8: Descripción CU: CII_CU1

... Trayectoria Principal ...

1. **La extensión** solicita al servidor autenticador registrar un nuevo usuario enviando los datos necesarios.
2. **El servidor autenticador** recibe los parámetros que le envió la extensión.
3. **El servidor autenticador** guarda al usuario en la base de datos.
4. **El servidor autenticador** genera un certificado de acuerdo a los datos dados por el usuario.
5. **El servidor autenticador** comunica a la extensión que el usuario ha sido creado.
6. **La extensión** despliega un mensaje de confirmación al usuario.

... Fin de la Trayectoria Principal ...

... Trayectoria alternativa 1 ...

1. **La extensión** solicita al servidor autenticador a registrar un nuevo usuario enviando los datos necesarios.

2. *El servidor autentificador* no recibe correctamente los datos.

3. *La extensión* despliega un mensaje de error.

... Fin de Trayectoria alternativa 1 ...

... Trayectoria alternativa 2 ...

1. *La extensión* solicita al servidor autentificador a registrar un nuevo usuario enviando los datos necesarios.

2. *El servidor autentificador* recibe los parámetros que le envió la extensión.

3. *El servidor autentificador* no puede registrar el nuevo usuario en su base de datos.

4. *La extensión* despliega un mensaje de error.

... Fin de Trayectoria alternativa 2 ...

Caso de uso: CII_CU2. Revocar certificado.	
Concepto	Descripción
Actor	Componente I. Extensión.
Propósito	Revocar el certificado que se encuentra vinculado a cierto usuario, cambiándolo por uno nuevo.
Entradas	Usuario y contraseña.
Salidas	Certificado nuevo.
Pre-condiciones	Usuario previamente registrado en el servidor autenticador.
Post-condiciones	-
Reglas del negocio	CII_RN1 CII_RN2 CII_RN4 CII_RN6
Errores	El certificado no se pudo crear correctamente. No se pudo asignar el certificado al usuario.

Cuadro 4.9: Descripción CU: CII_CU2

... Trayectoria principal ...

1. **La extensión** solicita revocar certificado al usuario actual enviando los datos necesarios.
2. **El servidor autenticador** valida si se encuentra registrado el usuario.
3. **El servidor autenticador** genera un nuevo certificado.
4. **El servidor autenticador** elimina el antiguo certificado asignado a ese usuario.
5. **El servidor autenticador** asigna el nuevo certificado al usuario.
6. **El servidor autenticador** envía un mensaje a la extensión de que se revocó correctamente el certificado.

... Fin Trayectoria principal ...

... Trayectoria Alternativa 1 ...

1. **La extensión** solicita revocar certificado al usuario actual.

2. *El servidor autentificador* no puede encontrar el usuario que solicitó revocar el certificado.
3. *El servidor autentificador* envía un código de error a la extensión.
4. *La extensión* despliega un mensaje de error.

... Fin Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. *La extensión* solicita revocar el certificado al usuario actual.
2. *El servidor autentificador* valida si se encuentra registrado el usuario.
3. *El servidor autentificador* no puede generar un nuevo certificado.
4. *El servidor autentificador* envía un código de error a la extensión.
5. *La extensión* despliega un mensaje de error.

... Fin Trayectoria Alternativa 2 ...

... Trayectoria Alternativa 3 ...

1. *La extensión* solicita revocar el certificado al usuario actual.
2. *El servidor autentificador* valida si se encuentra registrado el usuario.
3. *El servidor autentificador* genera un nuevo certificado.
4. *El servidor autentificador* elimina el antiguo certificado asignado a ese usuario.
5. *El servidor autentificador* asigna el certificado al usuario.
6. *El servidor autentificador* no puede enviarle el certificado al usuario.
7. *El servidor autentificador* envía un código de error a la extensión.
8. *La extensión* despliega un mensaje de error.

... Fin Trayectoria Alternativa 3 ...

Caso de uso: CII_CU3. Obtener certificado.	
Concepto	Descripción
Actor	Componente I. Extensión.
Propósito	Retornar el certificado del usuario a la extensión desde donde éste inició sesión.
Entradas	Usuario y contraseña.
Salidas	Certificado asignado a dicho usuario.
Pre-condiciones	CII_CU3.
Post-condiciones	-
Reglas del negocio	CII_RN1 CII_RN2 CII_RN3
Errores	El servidor autentificador no puede retornar el certificado a la extensión.

Cuadro 4.10: Descripción CU: CII_CU3

... Trayectoria principal ...

1. *La extensión* solicita el certificado asignado al usuario con sesión iniciada actualmente.
2. *El servidor autentificador* valida al usuario registrado en su base de datos.
3. *El servidor autentificador* envía a la extensión el certificado asociado a este usuario.

... Fin Trayectoria principal ...

... Trayectoria Alternativa 1 ...

1. *La extensión* solicita el certificado asignado al usuario con sesión iniciada actualmente.
2. *El servidor autentificador* no encuentra el usuario registrado en su base de datos.
3. *El servidor autentificador* envía un código de error a la extensión.
4. *La extensión* despliega un mensaje de error.

... Fin Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. *La extensión* solicita el certificado asignado al usuario con sesión iniciada actualmente.
2. *El servidor autenticador* valida el usuario registrado en su base de datos.
3. *El servidor autenticador* no puede enviar el certificado a la extensión.
4. *El servidor autenticador* envía un código de error a la extensión.
5. *La extensión* despliega un mensaje de error.

... Fin Trayectoria Alternativa 1 ...

Caso de uso: CII_CU4. Obtener código del certificado.	
Concepto	Descripción
Actor	Componente III. API.
Propósito	Retornar un SHA256 del certificado de un usuario especificado.
Entradas	Código SHA256 del nombre del usuario.
Salidas	Código SHA256 del certificado del usuario.
Pre-condiciones	Usuario registrado en el servidor autentificador.
Post-condiciones	-
Reglas del negocio	CII_RN1 CII_RN2 CII_RN6
Errores	El servidor autentificador no puede responder.

Cuadro 4.11: Descripción CU: CII_CU4

... Trayectoria principal ...

1. *El servidor autentificador* recibe un SHA256 del nombre de usuario enviado por **Componente III. API**.
2. *El servidor autentificador* busca el certificado de dicho usuario.
3. *El servidor autentificador* retorna un código SHA256 del certificado del usuario.

... Fin Trayectoria principal ...

... Trayectoria alternativa 1 ...

1. *El servidor autentificador* recibe un SHA256 del nombre de usuario enviado por **Componente III. API**.
2. *El servidor autentificador* busca el certificado de dicho usuario.
3. *El servidor autentificador* no encuentra el certificado debido a que el usuario no existe.
4. *El servidor autentificador* retorna un código de error.

... Fin Trayectoria alternativa 1 ...

4.2.2. Diagrama de flujo.

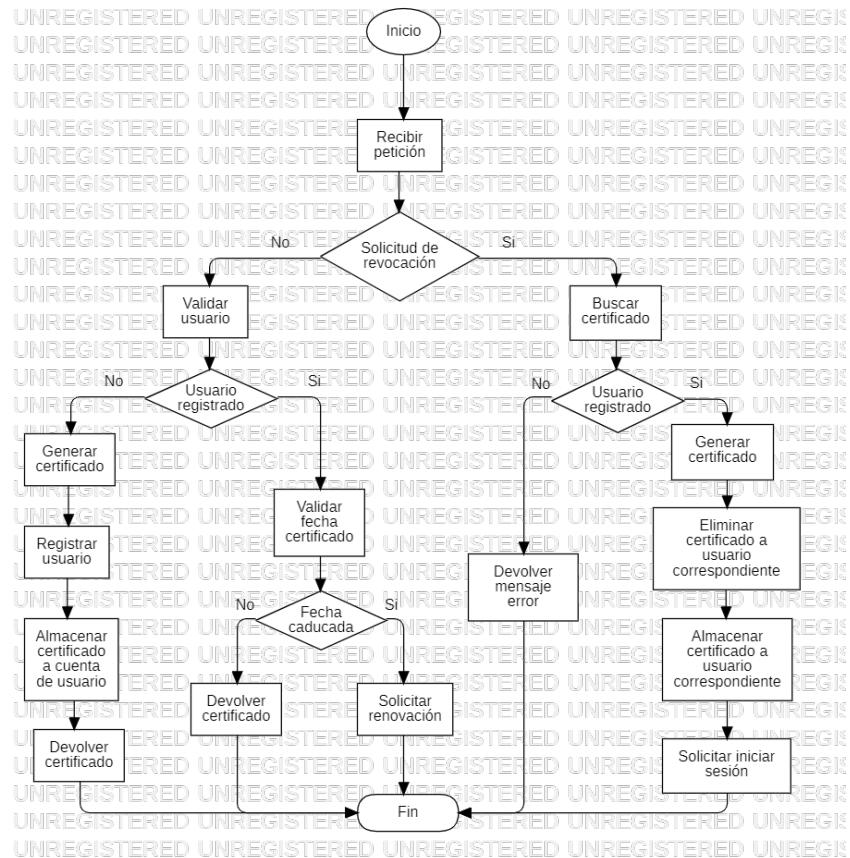


Figura 4.32: Diagrama de flujo de Componente II.

4.2.2.1. Descripción diagrama de flujo.

El flujo de este componente empieza al recibir una petición, posteriormente analiza que es lo que se le solicitó, ya que este componente tiene múltiples propósitos, principalmente registrar usuarios, verificar el certificado que recibe y revocar certificados ya existentes. El componente recibirá una petición y si esta es para el inicio de sesión, entonces el servidor autenticador validará al usuario, revisando si se encuentra registrado o no, para el primer caso validará que la fecha de expiración de dicho certificado siga vigente, si es así entonces devolverá el certificado de este usuario, si no se encuentra vigente entonces se le mandará un mensaje al usuario para solicitar la renovación del mismo. Por el otro lado si no se encuentra registrado, se debe de crear un

nuevo registro logrando de esta manera la generación de un certificado nuevo el cual se almacenará con los datos ingresados y finalmente se devolverá para que *la extensión* pueda hacer uso del mismo.

Existe también el caso en el que se solicite al servidor autenticador revocar un certificado, para ello se va a buscar dicho certificado y si se encuentra quiere decir que el usuario está registrado y se puede empezar el proceso de revocarlo, primero generando un certificado nuevo para este usuario, después se elimina el certificado asociado a este usuario, y se le asigna el certificado previamente creado a este nuevo usuario, por último se le manda un mensaje al usuario para que inicie sesión y se pueda devolver el nuevo certificado. En el caso en el que se pida revocar un certificado y no se encuentre ninguna cuenta registrada a este usuario entonces se mandará un mensaje de error.

4.2.3. Diagrama de flujo de datos.

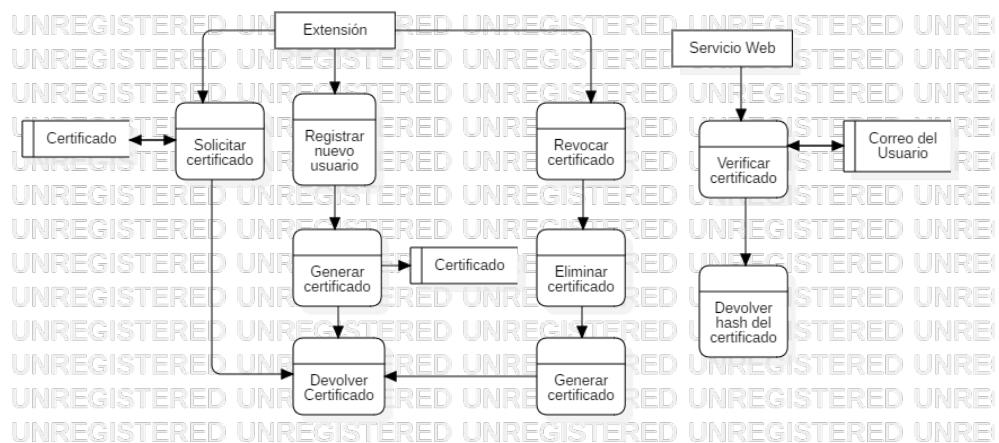


Figura 4.33: Diagrama de flujo de datos del Componente II.

4.2.3.1. Descripción diagrama de flujo de datos.

En este caso contamos con dos entidades externas, la extensión que por una parte puede solicitar un certificado mediante los datos solicitados por la autoridad, solicitar la generación de un nuevo usuario o bien solicitar la revocación de un certificado ligado a un usuario. La extensión le proporcionará los datos necesarios a la autoridad certificadora para que esta pueda crear y guardar al usuario con un certificado único para el mismo. Al solicitar un certificado la autoridad le devolverá a la extensión el certificado perteneciente al

usuario enviado. Si se solicita la creación de usuario, la autoridad solo devolverá como respuesta creación exitosa. Si la extensión solicita a la autoridad la revocación de un certificado, la autoridad necesita los datos del usuario a quien se le eliminará su certificado, para esto los datos del usuario deben ser correctos, si lo son, se procede a eliminar el certificado y a crear un nuevo, este certificado se guarda como certificado del usuario y se procede a devolver como respuesta dicho certificado creado previamente. Por otra parte, la API le solicitará a la autoridad la verificación de un certificado, esto con el fin de definir si el certificado fue generado por dicha autoridad y si ese certificado es valido.

4.2.4. Diagrama de clases.

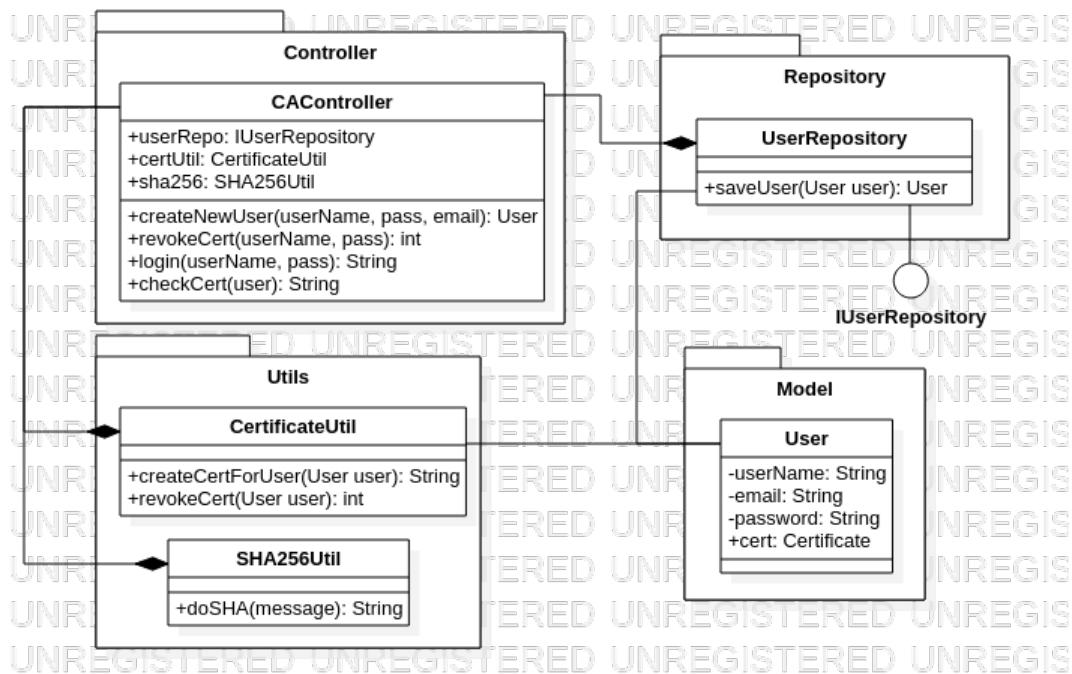


Figura 4.34: Diagrama de clases de Componente II.

4.2.4.1. Descripción de diagrama de clases

Clase: *CAController*

Atributos

1. **userRepo** : variable con la cual se pueden hacer operaciones con la base de datos.
 - Tipo de dato: **IUserRepository**.
2. **certUtil** : variable con la que se pueden hacer operaciones con y sobre los certificados, como crearlos o eliminarlos.
 - Tipo de dato: **CertificateUtil**.
3. **sha** : instancia para cifrar información con SHA256.
 - Tipo de dato: **SHA256Util**.

Métodos

1. **createNewUser(String pass, String email)**: Este método permite crear un nuevo usuario y guardar a éste en la base de datos .
 - Tipo de dato de retorno: **User**
2. **revokeCert(String email, String pass)**: Este método permite revocar el certificado actual del usuario con las credenciales recibidas.
 - Tipo de dato de retorno: **int**
3. **login(String pass, String email)**: Este método permite retornar el certificado del usuario de las credenciales recibidas.
 - Tipo de dato de retorno: **String**
4. **checkCert(String SHAuser)**: Este método permite retornar el sha256 del certificado del usuario recibido.
 - Tipo de dato de retorno: **String**

Clase: *CertificateUtil*

Métodos

1. **createCertForUser(User user)**: Este método permite crear un nuevo certificado con los datos del usuario.
 - Tipo de dato de retorno: **String**
2. **revokeCert(String userName, String pass)**: Este método permite revocar el certificado actual del usuario.

- Tipo de dato de retorno: **int**

Clase: *SHA256Util*

Métodos

1. **doSHA(String message)**: Este método permite cifrar el mensaje recibido con SHA256.

- Tipo de dato de retorno: **String**

Clase: *UserRepository*

Métodos

1. **saveUser(User user)**: Este método permite guardar en la base de datos el usuario recibido.

- Tipo de dato de retorno: **User**

Clase: *User*

Atributos

1. **email**: variable que almacena el email del usuario.

- Tipo de dato: **String**

2. **password**: variable que almacena la contraseña del usuario.

- Tipo de dato: **String**

3. **cert**: variable que almacena el certificado del usuario.

- Tipo de dato: **X509Certificate**

4.2.5. Diagramas de secuencias.

4.2.5.1. Diagrama de secuencia 1

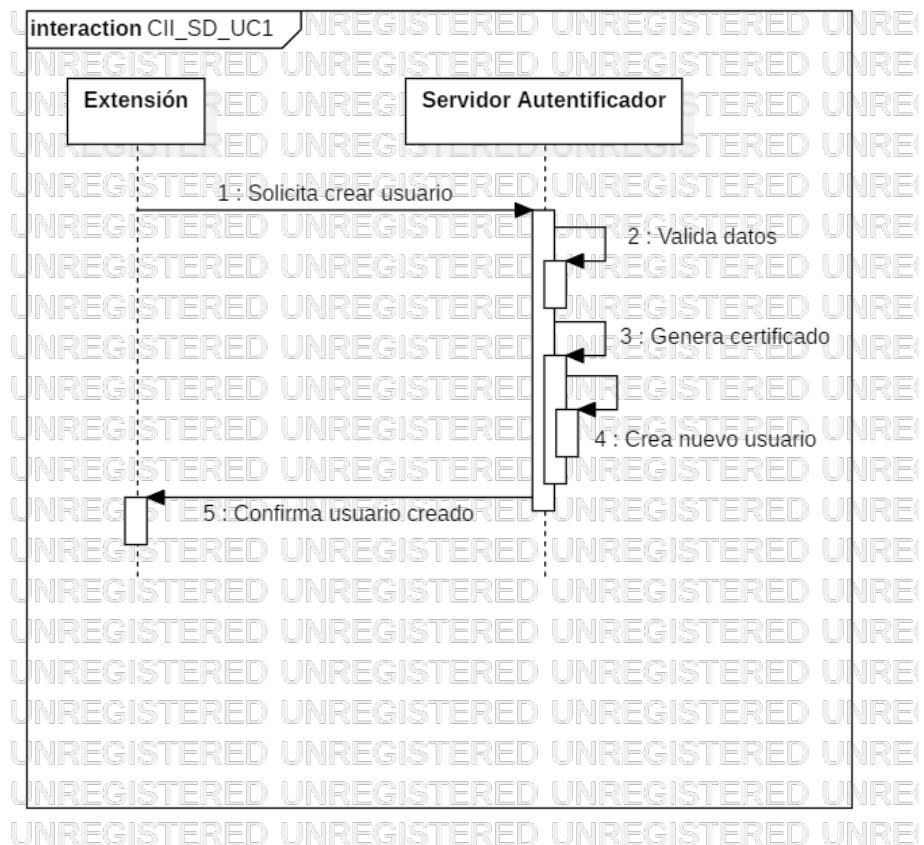


Figura 4.35: Diagrama de secuencia del CII_CU1. Crear Nuevo usuario.

Descripción: En este diagrama se explica el primer caso de uso, que es el de *crear un nuevo usuario*, donde el usuario solicita mediante la extensión el crear un nuevo usuario en el servidor autentificador, para que pueda empezar a utilizar nuestro servidor y asociar certificados a este usuario. Cabe resaltar que consideraremos cada correo que se reciba como un usuario.

4.2.5.2. Diagrama de secuencia 2

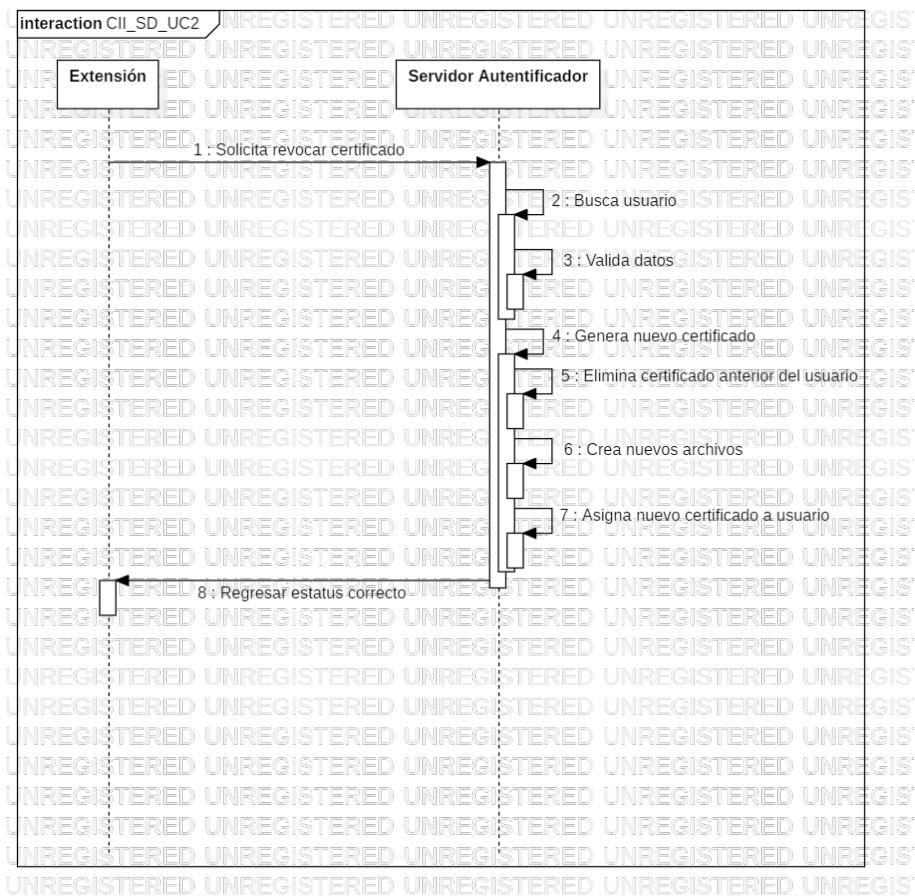


Figura 4.36: Diagrama de secuencia del CII_CU2. Revocar certificado

Descripción: En este diagrama, el usuario mediante la extensión solicita que se revoque su certificado, el servidor recibe esta solicitud y valida los datos, si son correctos genera un nuevo certificado y se lo asigna a este usuario.

4.2.5.3. Diagrama de secuencia 3

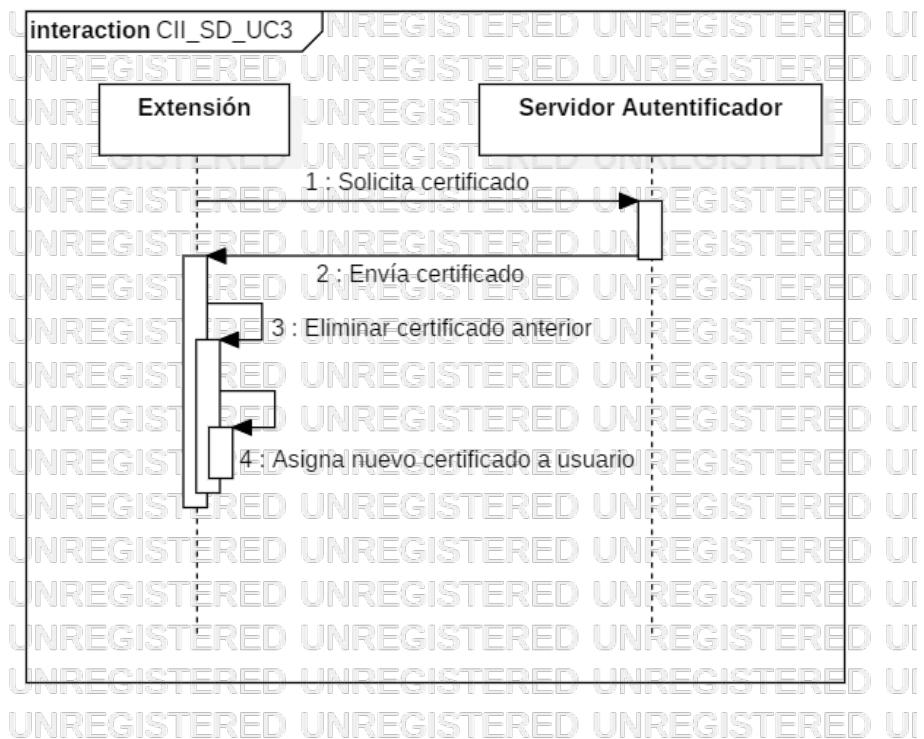


Figura 4.37: Diagrama de secuencia del CII_CU3. Obtener certificado.

Descripción: Para este diagrama, se obtienen un certificado desde el servidor a la extensión, esta después valida los datos y si son correctos, guarda en el storage de Chrome el certificado de ese usuario.

4.2.5.4. Diagrama de secuencia 4

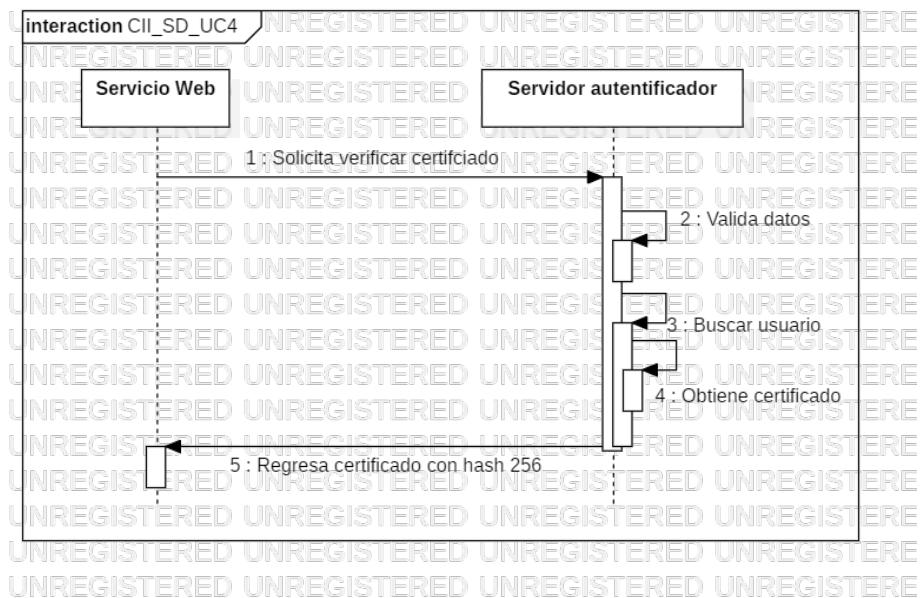


Figura 4.38: Diagrama de secuencia del CII_CU4. Verificar certificado.

Descripción: Este diagrama servira para verificar los certificados que obtiene el servidor desde la extensión, donde el servidor valida si el certificado que recibe existe en su base de datos, y le envía una respuesta a la extensión, en este caso se regresa como respuesta un hash 256 del certificado del usuario..

4.2.6. Diagrama de actividades.

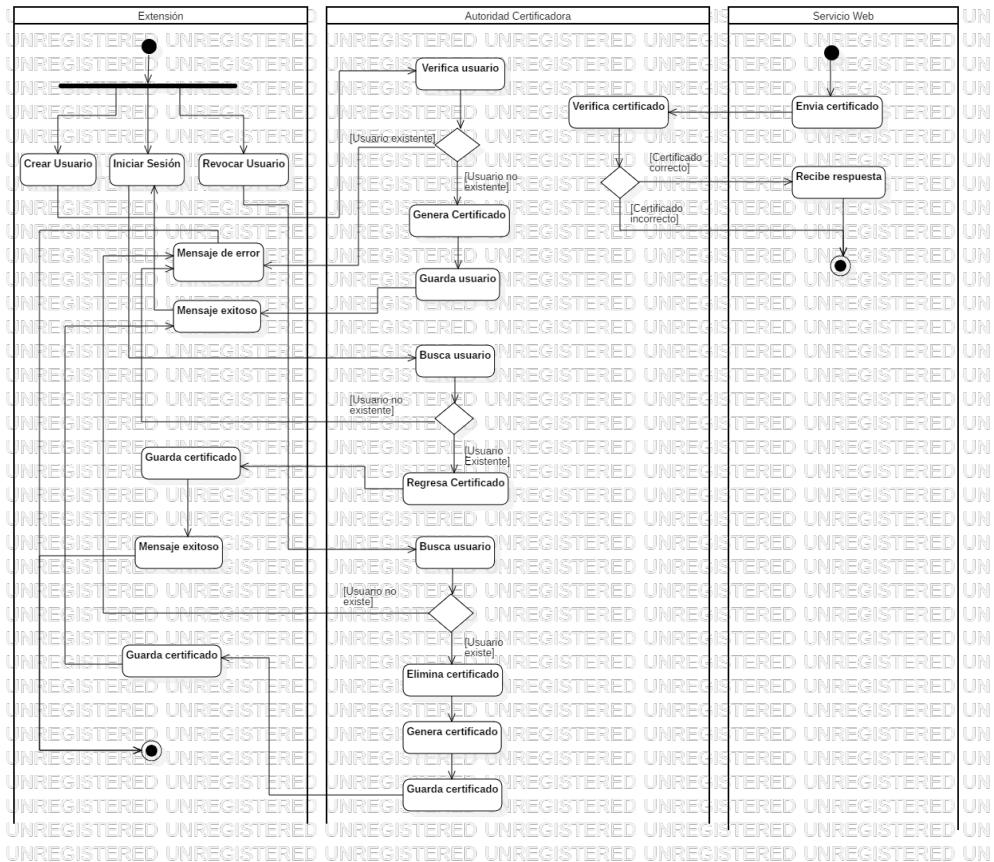


Figura 4.39: Diagrama de actividades de Componente II.

4.2.6.1. Descripción del diagrama de actividades.

El componente cuenta con los siguientes pasos. Por una parte la interacción entre la extensión y la autoridad certificadora, la cual la extensión podrá solicitar la creación de un nuevo usuario, iniciar sesión para obtener el certificado y la revocación de un certificado. Para el inicio de sesión la extensión debe de enviarle los datos de usuario para así la autoridad poder verificar dichos datos, si los datos son correctos y existen en la base de datos se procede a generar un certificado para el usuario y se registra en la base de datos. Si se solicita el inicio de sesión, es decir obtener un certificado, la autoridad al recibir dicha petición se procede a buscar el usuario solicitado, si existe la autoridad regresa como respuesta el certificado del usuario. Si la extensión solicita la revocación de un certificado, la autoridad debe buscar al

usuario en la base de datos, si existe un usuario con esos datos, se procede a eliminar su certificado y crear uno nuevo, devolviendo como respuesta el nuevo certificado. Ahora bien, si la API solicita la verificación de un certificado, la autoridad deberá verificar dicho certificado, si el certificado fue creado por la autoridad, este regresa respuesta satisfactoria.

4.3. Componente III: API.

4.3.1. Diagrama de casos de uso.

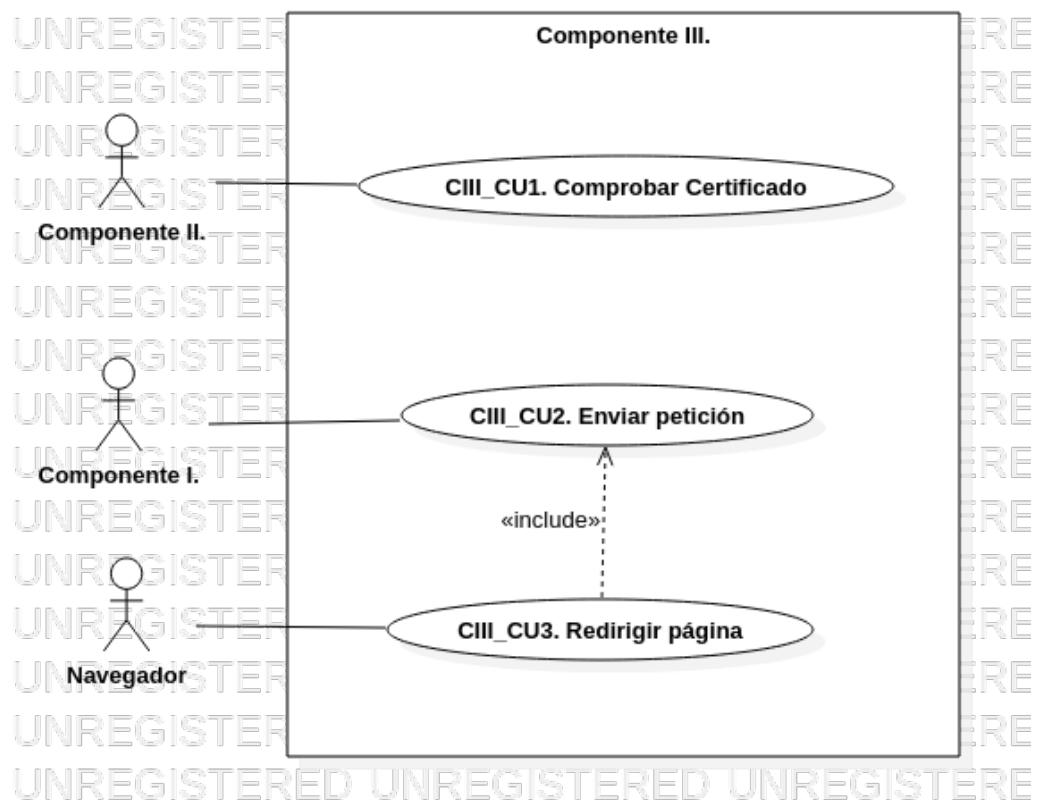


Figura 4.40: Diagrama de caso de uso del componente III

4.3.1.1. Descripción de diagrama de casos de uso.

Caso de uso: CIII_CU1. Comprobar certificado.	
Concepto	Descripción
Actor	Componente II. Servidor autentificador
Propósito	Saber si el certificado ha sido o no revocado por el usuario.
Entradas	SHA256 del usuario del certificado.
Salidas	SHA256 del certificado en el servidor autentificador del usuario.
Pre-condiciones	-
Post-condiciones	-
Reglas del negocio	CIII_RN3
Errores	No se encuentra al usuario.

Cuadro 4.12: Descripción CU: CIII_CU1

... Trayectoria Principal ...

1. *La API* envía un SHA256 del usuario del certificado al *Servidor autentificador*.
2. *El servidor autentificador* retorna un SHA256 del certificado actual en el servidor de ese usuario.
3. *La API* comprueba que el SHA256 del certificado obtenido de la petición es igual al SHA256 del certificado obtenido del *Servidor autentificador*
4. *La API* retorna al *Servicio web* que el certificado es válido

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. *La API* envía un SHA256 del usuario del certificado al *Servidor autentificador*.
2. *El servidor autentificador* retorna un SHA256 del certificado actual en el servidor de ese usuario.

3. ***La API*** comprueba que el SHA256 del certificado obtenido de la petición no es igual al SHA256 del certificado obtenido del ***Servidor autenticador***
4. ***La API*** retorna al ***Servicio web*** que el certificado es inválido

... Fin de la Trayectoria Alternativa 1 ...

... Trayectoria Alternativa 2 ...

1. ***La API*** envía un SHA256 del usuario del certificado al ***Servidor autenticador***.
2. ***El servidor autenticador*** no encuantra al usuario y retorna un código de error.
3. ***La API*** retorna al ***Servicio web*** que el certificado es inválido

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CIII_CU2. Enviar petición.	
Concepto	Descripción
Actor	Componente I. Extensión
Propósito	Recibir la petición HTTP con el certificado inyectado.
Entradas	Petición HTTP con certificado inyectado para poder iniciar sesión.
Salidas	Acceso o no para el usuario al servicio web.
Pre-condiciones	-.
Post-condiciones	CIII CU4
Reglas del negocio	CIII RN4
Errores	La petición no tiene el certificado inyectado.

Cuadro 4.13: Descripción CU: CIII_CU2

... Trayectoria Principal ...

1. ***La extensión*** envía el certificado inyectado en la petición HTTP
2. ***El servicio web*** recibe la petición.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. ***La extensión*** no envía el certificado inyectado en la petición HTTP
2. ***El servicio web*** recibe la petición.

... Fin de la Trayectoria Alternativa 1 ...

Caso de uso: CIII_CU3. Redirigir página.	
Concepto	Descripción
Actor	Navegador
Propósito	Redirigir la respuesta del servicio web para darle información al usuario acerca de su inicio de sesión.
Entradas	Respuesta del servicio web.
Salidas	Despliegue de la respuesta en el navegador.
Pre-condiciones	CIII_CU3.
Post-condiciones	-
Reglas del negocio	-
Errores	No se puede desplegar la respuesta del servicio web.

Cuadro 4.14: Descripción CU: CIII_CU3

... Trayectoria Principal ...

1. *El servicio web* envía la respuesta.
2. *El navegador* muestra la respuesta.

... Fin de la Trayectoria Principal ...

... Trayectoria Alternativa 1 ...

1. *El servicio web* envía la respuesta.
2. *El navegador* no muestra la respuesta.

... Fin de la Trayectoria Alternativa 1 ...

4.3.2. Diagrama de flujo.

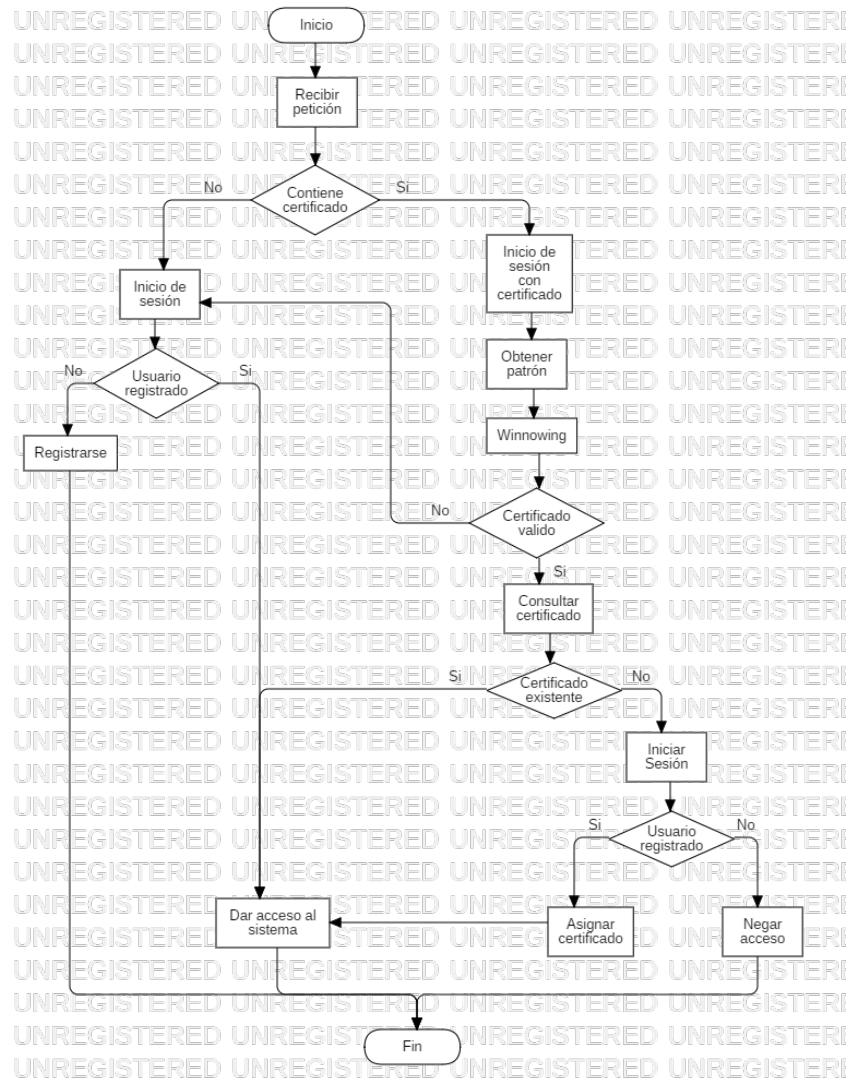


Figura 4.41: Diagrama de flujo de Componente III.

4.3.2.1. Descripción diagrama de flujo.

Para el caso de este diagrama se inicia con una petición, la cual es recibida por la API y analizada para verificar si el encabezado tiene el certificado. Si la petición no tiene el certificado, este componente ignora dicha petición. Si la petición contiene un certificado, se procede a iniciar sesión con certificado,

Si se realiza satisfactoriamente el winnowing, se inicia sesión donde puede o no estar registrado el usuario, de cualquier manera el usuario al registrarse o iniciar sesión se usara el certificado.

4.3.3. Diagrama de flujo de datos.

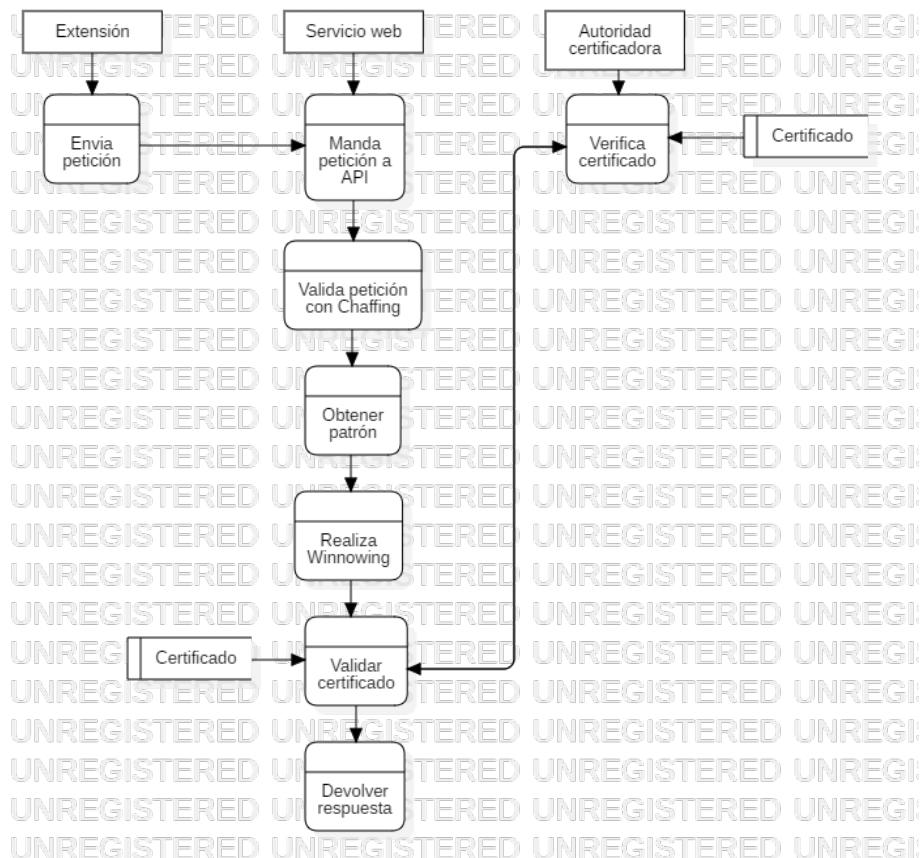


Figura 4.42: Diagrama de flujo de datos de Componente III.

4.3.3.1. Descripción diagrama de flujo de datos.

A continuación tenemos el diagrama de flujo de datos, donde podemos ver cláaramente como viaja la información principal a traves de este componente y con las entidades externas, primero la extensión envía una petición al servicio web, este lo recibe, la API lo intercepta y verifica si es una petición con Chaffing que debe de ser analizada, si es así realiza la etapa de winnowing descifrando el patrón con su llave privada y por último obtiene el certificado

dentro de esta petición y la compara con las que cuenta con el servicio web, para saber si debe de dar una respuesta de usuario o solicitar que inicie sesión en este mismo.

4.3.4. Diagrama de clases.

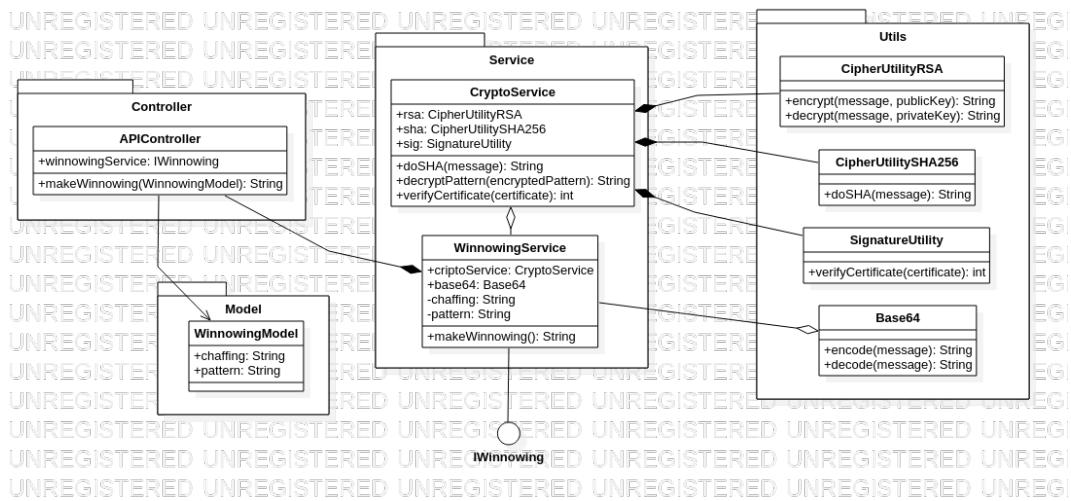


Figura 4.43: Diagrama de Clases de componente 3.

4.3.4.1. Descripción diagrama de clases.

Clase: *APIController*

Atributos

1. **winnowingService** : variable donde se tendrá acceso a los métodos necesarios para realizar la etapa de winnowing.
 - Tipo de dato: **IWInnowing**.

Métodos

1. **makeWinnowing(WinnowingModel wm)**: Este método realiza la etapa de winnowing, retorna el certificado.
 - Tipo de dato de retorno: **String**

Clase: *WinnowingModel*

Atributos

1. **chaffing** : variable donde se tiene guardado el chaffing de la petición.
 - Tipo de dato: **String**.
2. **pattern** : variable donde se tiene guardado el patrón de la petición.
 - Tipo de dato: **String**.

Clase: *CryptoService*

Atributos

1. **rsa** : instancia para descifrar información con RSA.
 - Tipo de dato: **CipherUtilityRSA**.
2. **sha** : instancia para cifrar información con SHA256.
 - Tipo de dato: **CipherUtilitySHA256**.
3. **sig** : instancia para validar información de un certificado.
 - Tipo de dato: **SignatureUtility**.

Métodos

1. **doSHA(String message)**: Este método calcula el sha256 de la cadena de texto message.
 - Tipo de dato de retorno: **String**
2. **decryptPattern(String encryptedPattern)**: Este método descifra el patrón de chaffing.
 - Tipo de dato de retorno: **String**
3. **verifyCertificate(String certificate)**: Este método verifica la autenticidad de un certificado.
 - Tipo de dato de retorno: **int**

Clase: *WinnowingService*

Atributos

1. **cryptoService** : instancia para acceder a todas las utilidades de cifrado.
 - Tipo de dato: **CryptoService**.
2. **base64** : instancia para decodificar información en formato BASE64.
 - Tipo de dato: **Base64**.
3. **chaffing** : variable para guardar el chaffing actual.
 - Tipo de dato: **String**.
4. **pattern** : variable para guardar el patron de chaffing actual.
 - Tipo de dato: **String**.

Métodos

1. **makeWinnowing()**: Este método realiza la etapa de winnowing.
 - Tipo de dato de retorno: **String**

Clase: *CipherUtilityRSA*

Métodos

1. **encrypt(String message, PublicKey publicKey)**: Este método cifra un mensaje con la llave pública especificada.
 - Tipo de dato de retorno: **String**
2. **decrypt(String message, PrivateKey privateKey)**: Este método descifra un mensaje con la llave privada especificada.
 - Tipo de dato de retorno: **String**

Clase: *CipherUtilitySHA256*

Métodos

1. **doSHA(String message)**: Este método cifra el mensaje que se le manda.
 - Tipo de dato de retorno: **String**

Clase: *Signature Utility*

Métodos

1. **verifyCertificado(String certificate)**: Este método verifica la validez de un certificado.
 - Tipo de dato de retorno: **int**

Clase: *Base64*

Métodos

1. **encode(String message)**: Este método codifica el mensaje recibido a base64.
 - Tipo de dato de retorno: **String**
2. **decode(String message)**: Este método decodifica el mensaje recibido de base64 a UTF-8.
 - Tipo de dato de retorno: **String**

4.3.5. Diagramas de secuencias.

Diagrama de Secuencia 1. Comprobar Certificado.

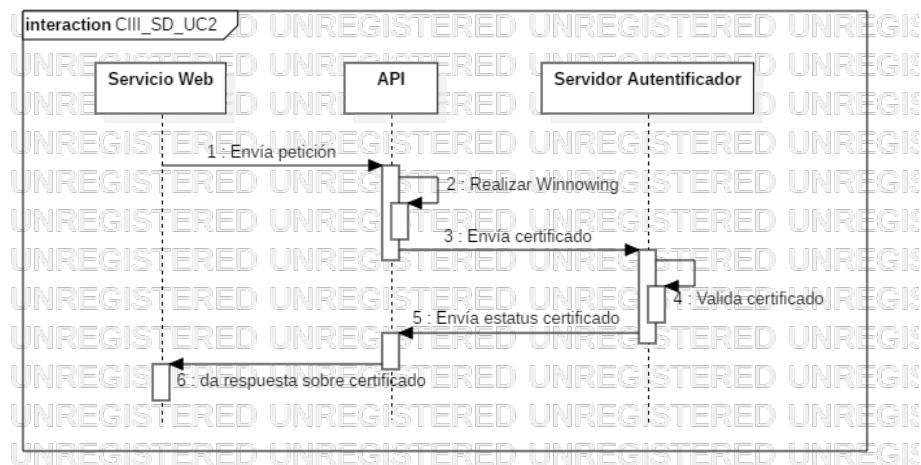


Figura 4.44: Diagrama de Secuencia de Caso de Uso 1. Comprobar certificado.

4.3.5.1. Descripción de diagrama de Secuencia CIII_SD_UC1.

Después de que el servicio web reciba una petición, la API la interceptará para realizar la etapa de Winnowing y posteriormente envíe el certificado al servidor autenticador, el Servidor Autentificador validará los datos del usuario y dependiendo si existe o no ese usuario registrado, con un certificado válido(actualizado) o uno inválido(revocado) le regresará un status a la API y está sabrá el como darle respuesta al Servicio Web basado en el estatus que recibió del Servidor Autentificador.

Diagrama de Secuencia 2. Enviar Petición.

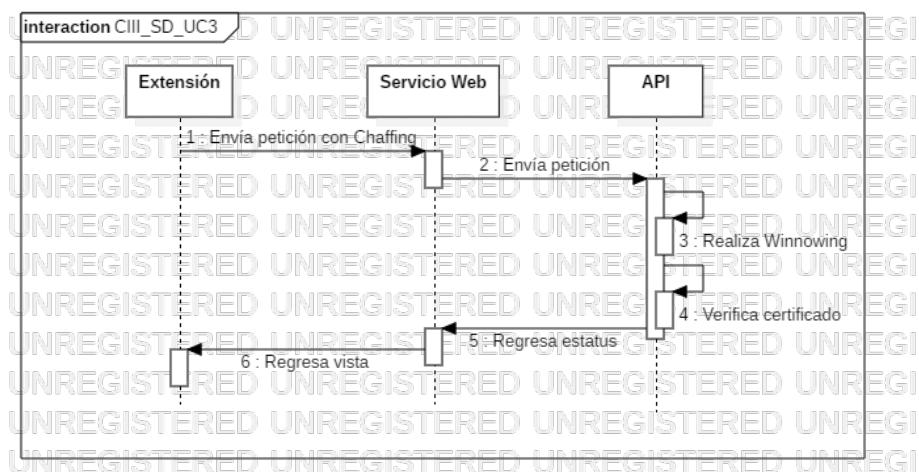


Figura 4.45: Diagrama de Secuencia de Caso de Uso 2. Enviar petición.

4.3.5.2. Descripción de diagrama de Secuencia CIII_SD_UC2.

La extensión envía una petición con Chaffing al Servicio Web, donde la API intercepta dicha petición para analizar si es una petición que contenga un Chaffing el cuál podamos analizar, posteriormente realizará la etapa de Winnowing si es una petición de nuestro interés y enviará el certificado al servicio web.

Diagrama de Secuencia 3. Redirigir Página.

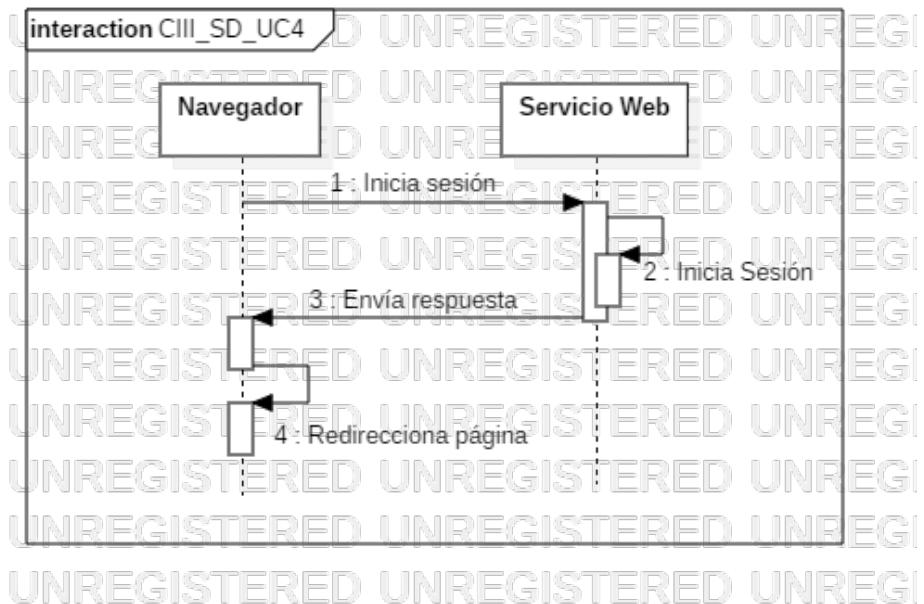


Figura 4.46: Diagrama de Secuencia de Caso de Uso 3. Redirigir página.

4.3.5.3. Descripción de diagrama de Secuencia CIII_SD_UC3.

El navegador teniendo lista la petición con Chaffing la envía al Servicio Web, donde éste mediante la API, valida dicha petición y elige el inicio de sesión para el usuario, posteriormente le envía una respuesta al navegador y por último este mismo redirige la página dependiendo es esta misma respuesta.

4.3.6. Diagrama de actividades

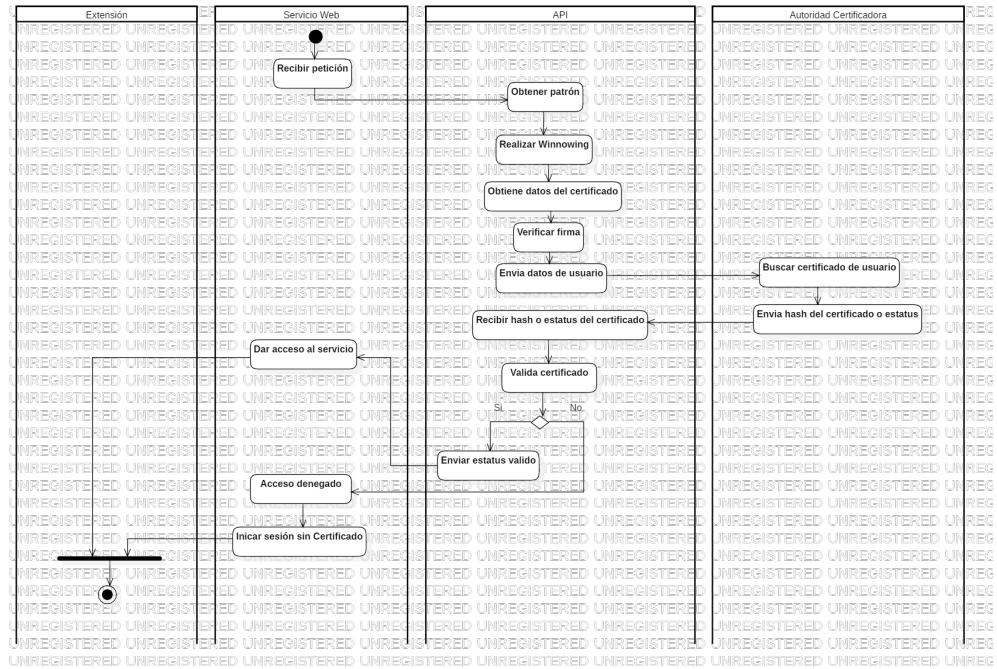


Figura 4.47: Diagrama de Actividades de componente 3.

4.3.6.1. Descripción diagrama de actividades.

Cuando el servicio web recibe una petición con chaffing, realiza la etapa de Winnowing para poder obtener el certificado, y posteriormente se comunica con la autoridad certificadora, este comparará el certificado, ya que existen 3 casos posibles: el primero es que el certificado sea válido, y simplemente le dará acceso al servicio web con los datos del certificado correspondiente, el segundo caso es que el usuario sea válido, pero el certificado haya sido revocado antes y necesite actualizar su certificado en ese ordenador y el tercero es que el certificado sea inválido, por lo cual le dará un mensaje al usuario de que el certificado no es el correcto.

4.3.7. Interfaz de usuario.

Una vez que el usuario tiene una cuenta y ha obtenido su certificado de la autoridad certificadora. El usuario puede proceder a navegar en la red para hacer uso de la extensión. En el componente 3, las interfaces que se muestran al usuario son las vistas del servicio web que se ha implementado para llevar

a cabo el proceso de *Winnowing*.

El servicio web de prueba que se utilizará para este trabajo es una pagina web para usuarios y doctores de una *veterinaria*, la cual se utilizará para la modificación correspondiente donde el servicio lleve a cabo la autenticación por *Chaffing and Winnowing*.

El servicio web de la veterinaria se muestra en la figura 4.48.



Figura 4.48: Interfaz principal del servicio web de veterinaria.

En esta interfaz dada por el servicio web, le permite al usuario iniciar sesión. Como la extensión esta activada, ésta realizará la tarea de bloquear la petición, para hacer el proceso de *Chaffing* y posteriormente inyectar el resultado en el encabezado de la petición (ver Componente 1).

Al recibir la petición en el servicio web, éste enviará la petición a la API que se implementa en el servicio web. La API es la encargada de realizar el proceso de *Winnowing* y verificará la autenticidad del certificado.

Una vez verificada la autenticidad del certificado, se muestra la interfaz de la figura 4.49 en el cual se le da a indicar al usuario que se ha detectado un certificado y esta listo para vincularlo a una cuenta existente en el servicio web.



Figura 4.49: Interfaz para iniciar sesión en el servicio web con certificado en el encabezado de la petición.

Una vez que el usuario ingrese sus credenciales correctas del servicio web, éste vinculará el certificado a este usuario y a su vez le dará acceso al servicio con su cuenta. La figura 4.50 muestra la página de inicio del servicio web con la cuenta del usuario la cual ya tiene un certificado vinculado.

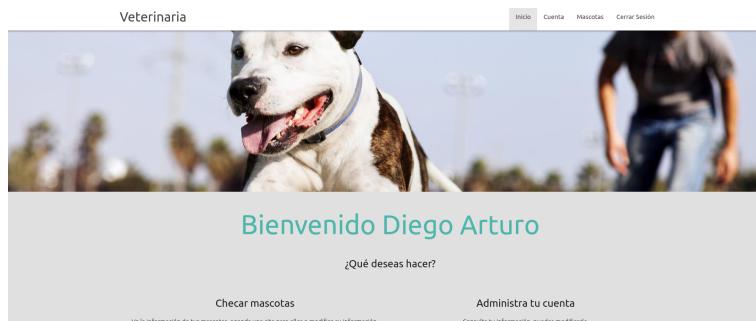


Figura 4.50: Interfaz del servicio web con acceso de una cuenta.

El servicio web le permite al usuario crear una nueva cuenta en el servicio, ya que es necesario para poder vincular el certificado. La figura 4.51 muestra la interfaz que se le presenta al usuario para la creación de una nueva cuenta en el servicio web.

Veterinaria

Iniciar Sesión Registrar

Completa el siguiente formulario

Información general

Nombre _____

Primer apellido _____

Segundo apellido _____

E-mail _____

Día de nacimiento Elige ▾

Mes de nacimiento Elige ▾

Año de nacimiento Elige ▾

Información de la cuenta

Soy veterinario

Figura 4.51: Interfaz del servicio web para crear nuevo usuario.

Otro caso a considerar es cuando el usuario no activa la extensión y quiere ingresar al servicio web, por lo que si el usuario realiza una petición al servicio con la extensión deshabilitada, el servicio web no encontrará ningún certificado con *Chaffing* inyectado en el encabezado, por lo que el servicio web pedirá realizar un inicio de sesión ordinario, el cual le pedirá usuario y contraseña para ingresar al servicio con su cuenta.

La figura 4.52 muestra la interfaz que se le muestra al usuario cuando se requiere un inicio de sesión ordinario.

Inicia sesión

Nombre de Usuario

Contraseña de Usuario

INICIAR SESIÓN >

Figura 4.52: Interfaz de inicio de sesión ordinario del servicio web.

Como sabemos, el usuario puede revocar un certificado lo cual quiere decir que puede generar uno nuevo. Esto le permite al usuario poder generar un

nuevo certificado si el usuario dejo su sesión iniciada en otra maquina publica. El tener la capacidad de generar otro certificado, le permite al servicio web no dar acceso si el certificado recibido se encuentra revocado, pidiendo asi iniciar sesión de nuevo con credenciales (usuario y contraseña) para vincular el nuevo certificado. La figura 4.53 es la interfaz que se le muestra al usuario si el certificado que el servicio web obtiene del encabezado se encuentra revocado.



Figura 4.53: Mensaje de inicio de sesión incorrecto por revocación de certificado ó credenciales inválidas.

4.3.8. Algoritmos.

Los algoritmos necesarios para hacer el *winnowing* así como la verificación de los certificados son expuestos a continuación:

```

Data: chaffing, patternCipher, aesCipher, privateKey
Result: cert,header
1 chaffingDecode[] ← base64.decode(chaffing);
2 aesKey ← rsa.decrypt(aesCipher,privateKey);
3 pattern[] ← aes.decrypt(patternCipher,aesKey);
4 cert[];
5 header[];
6 i ← 0;
7 while i < pattern.length do
8   | if pattern[i] == 1 then
9     |   | header.add(chaffingDecode[i]);
10    | else
11      |   | cert.add(chaffingDecode[i]);
12    | end
13    | i ← i + 1;
14 end

```

Algoritmo 3: Obtención de la cabecera y certificado mediante el proceso de winnowing.

En este algoritmo se toman en consideración los siguientes datos de la petición HTTP recibida:

- **chaffing**: es la cadena que se encuentra en base 64 que contiene el certificado y la basura mezclada.
- **aesCipher**: Se encuentra en el mismo campo del patrón, contiene la llave AES cifrada mediante el cifrado asimétrico RSA.
- **patternCipher**: es el otro apartado que contiene el patrón necesario para realizar el proceso de winnowing, el cual está cifrado con el algoritmo de cifrado simétrico AES.

Así como la llave privada necesaria para realizar el decifrado RSA **privateKey** la cual si bien no se envía en la petición, se encuentra almacenada localmente como dato estático. El proceso de winnowing se realiza analizando el patrón para dividir cada uno de los bits del chaffing en certificado y cabecera.

En cuanto a la salida de este algoritmo se hace mención de dos datos los cuales se despliegan a continuación:

- **cert**: Contiene el certificado del usuario completamente íntegro asumiendo que el proceso fue correcto.

- **header:** Contiene la información del campo *Accept* de la cabecera HTTP.

Por otro lado se cuenta con otro algoritmo para realizar la verificación del certificado buscando que no se incluya algún certificado no válido o que sea expedido por alguna AC externa a la desarrollada por nosotros:

```

Data: cert,publicKey
Result: certR,flag
1 if cert != null and cert.verify(publicKey) == 1 then
2   | dataCert ← getDataCert(cert);
3   | emailUser ← dataCert.getEmail();
4   | response ← CA.getValidation(sha256.doSha(emailUser));
5   | shaCert ← sha256.doSha(cert);
6   | if response == 0 then
7     |   | certR = 0;
8     |   | flag = 0;
9   | else
10    |   | if response == shaCert then
11      |   |   | certR = cert;
12      |   |   | flag = 1;
13    |   | else
14      |   |   | certR = 0;
15      |   |   | flag = 2;
16    |   | end
17  | end
18 else
19   |   | certR = 0;
20   |   | flag = 0;
21 end
```

Algoritmo 4: Algoritmo para la verificación del certificado.

4.3.8.1. Complejidad computacional.

Haciendo un análisis del algoritmo de winnowing anteriormente mostrado, deducimos que la complejidad del algoritmo de *winnowing* es: $O(n)$ donde n es el tamaño de caracteres del certificado.

Para el caso de la verificación se cuenta con los siguientes datos de entrada:

- **cert:** Contiene el certificado recibido en la petición HTTP proveniente del anterior algoritmo.

- **publicKey:** Contiene la public Key de la AC necesaria para la comprobación del certificado.

También es importante dejar en claro algunas funciones con las que se cuenta en el algoritmo como es el caso de **getDataCert(key)** que es el algoritmo que se encarga de comprobar tanto la fecha del certificado como validar que el mismo haya sido expedido por la AC correspondiente, por otro lado la función **CA.getValidation(sha256)** es una función que se encarga de comunicarse con la AC enviándole un sha del usuario, que en este caso es el email, para que busque en sus repositorios si el certificado aún se encuentra activo para el nombre de usuario dado, a lo que nos responderá *0* si el usuario no existe o *ShaCertificadoUsuario* si el usuario cuenta con un certificado válido para realizar una comparación con el recibido de la petición. La salida de este algoritmo se compone de lo siguiente:

- **certR:** Contiene el certificado después de validarse para que el servicio web pueda almacenarlo en su base de datos.
- **flag:** Contiene una bandera de control para saber el resultado de la validación cuyos valores posibles son los siguientes:
 - *0*: Se refiere a que el certificado recibido no es válido.
 - *1*: Se refiere a que el certificado recibido es válido.
 - *2*: Se refiere a que el certificado recibido es válido pero ya no se encuentra asignado al usuario en cuestión, es decir, fue revocado.

Por lo que de esta manera el servicio web puede decidir qué hacer en cada uno de los casos, tomando en cuenta la recomendación de no mostrar demasiada información al usuario cuidando siempre la seguridad de los sistemas.

Capítulo 5

Desarrollo.

5.1. Componente I. Extensión.

En esta sección se explicará de manera breve y puntual los pasos que se siguieron para la implementación de la extensión de Google Chrome, así como las funciones que interactúan para que logre cada una de sus funciones. Antes que nada al iniciar una extensión es necesario contar con un archivo llamado **manifest.json** el cual contendrá información básica de nuestra extensión así como los scripts y recursos que se necesitan para su correcto funcionamiento, el cual se menciona a continuación:

```
1 {
2   "name": "TT_Project",
3   "version": "1.0",
4   "manifest_version": 2,
5   "description": "Keep safe your passwords",
6   "icons": {
7     "128": "img/escom.png"
8   },
9   "browser_action": {
10     "default_icon": "img/escom.png",
11     "default_popup": "popup.html",
12     "default_title": "TT_Project_ESCOM"
13   },
14   "background": {
15     "scripts": ["js/jQuery/jquery-3.3.1.js", "js/
CryptoJs/CryptoJs.js", "js/rsa/jsencrypt.min.js", "background.js"],
16     "persistent": true
}
```

```

17 },
18 "minimum_chrome_version": "18",
19 "permissions": [ "webRequest", "webRequestBlocking",
20 " ", "<all_urls>", "storage", "tabs" ],
21 "content_security_policy": "script-src 'self' "
22 "unsafe-eval"; object-src 'self' "
}

```

Código 5.1: manifest.json

De esta manera el navegador conoce el ícono de la extensión, la página del popup, los scripts, los permisos entre otros. de esa forma la extensión tiene la estructura siguiente:

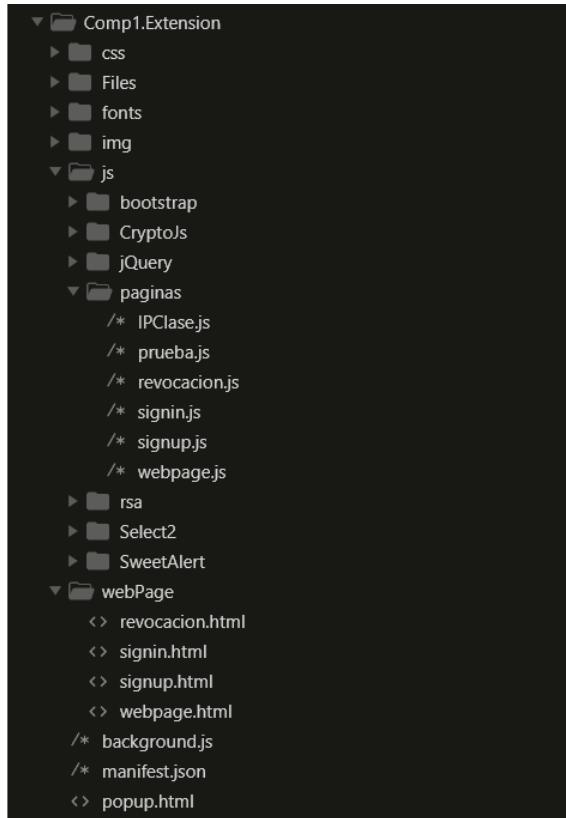


Figura 5.1: Estructura del desarrollo de la Extensión.

Como podemos ver hay dos archivos que acompañan al **manifest.json** llamados **popup.html** y **background.js** de los cuales hablaremos a contin-

nuación.

5.1.1. Archivo popup.html

Es un HTML que contiene la página que se mostrará al hacer click en el botón de la extensión, la idea de esta *mini página* es que nos permita gestionar las funciones de la extensión, por tal motivo contará con 4 botones necesarios para su funcionamiento:

- *btnActivar* : Botón necesario para activar o desactivar el funcionamiento de la extensión.
- *btnIniciarSesion* : Botón en el que el usuario podrá iniciar sesión o registrarte para empezar a utilizar la extensión.
- *btnCerrarSesion* : Botón que nos permite cerrar la sesión del usuario actual.
- *btnAviso* : Botón que nos redirige a una página en donde podremos aceptar el certificado, así como muestra algunos datos de cómo funciona el proceso de autentificación.

5.1.2. Archivo background.js

Es el archivo que contiene la mayor parte de la lógica del comportamiento de este componente, vamos a explicar alguna de las funciones más relevantes de su desarrollo

Lo primero que debe de realizar la extensión es interceptar la petición antes de que esta salga a red para poder realizar todo el proceso de Chaffing.

5.1.2.1. Interceptar petición.

Primero que nada, se necesita comprobar que la extensión se encuentre habilitada, checando la bandera de *btnActivar*, si se encuentra en *true* entonces procedemos a obtener todos los parametros que necesitaremos manejar.

```
1 chrome.webRequest.onBeforeSendHeaders.addListener(  
2   function(details){  
3     // Función del API de Google Chrome que obtiene el valor  
4     // guardado en el Storage  
5     chrome.storage.local.get(['Activo'],function(result){  
6       if(result.Activo == null){  
7         btnActivar = true;
```

```

7     }else{
8         btnActivar = result.Activo;
9     }
10    });
11
12    if (btnActivar == true){
13        let detailsComplete = details;
14        let requestid = details.requestId;
15        let method = details.method;
16        let frameid = details.frameId;
17        let parentframeid = details.parentFrameId;
18        let tabid = details.tabId;
19        let initiator = details.initiator;
20        let timestamp = details.timeStamp;
21        let url = details.url;
22        let httpheaders = details.requestHeaders;
23        let tipo = details.type;

```

Código 5.2: background.js función para interceptar datos del encabezado

Una vez que ya tenemos todos nuestros datos entonces vamos a filtrar el tipo de petición, ya que para este prototipo nos interesa únicamente la petición de tipo *main_frame* y obteniendo esta petición vamos a obtener el header de *Accept*, con esto puede realizar el llamado a la función de *getCertificateFromStorage*.

```

1     if(tipo.includes("main_frame")){
2
3         let cabeceraInyectar = "";
4
5         for(let i = 0; i < httpheaders.length; i++){
6             if(httpheaders[i].name.includes("Accept")){
7                 cabeceraInyectar = httpheaders[i].value;
8             }
9         }
10        getCertificateFromStorage(cabeceraInyectar ,
11        detailsComplete);
12        //Bloquea petición
13        return {cancel: true};
14    }

```

Código 5.3: background.js Obteniendo el header de accept y bloqueando la petición

Finalmente se bloquea la petición para poder realizar los procesos que siguen a continuación.

5.1.2.2. Creación del patrón de Chaffing

Esta función nos servirá para poder generar el patrón de chaffing de manera aleatoria, utilizando la lógica que se menciona en el resto del documento, la función de getSecureRandomNumber coloca unos en alguna posición de nuestro patrón igual al número de caracteres que tenga el certificado, recibido en esta misma función.

```
1 function getPatternBite(LenCertificadoCharArray ,  
2   LenCabeceraInyectar){  
3  
4   let lengthPc = (LenCabeceraInyectar+LenCertificadoCharArray  
5     )*8;  
6  
7   let pcArray = new Array(lengthPc);  
8   pcArray.fill(1,0,lengthPc);  
9  
10  let n_0 = 0;  
11  while(n_0 < LenCertificadoCharArray*8){  
12    let random = getSecureRandomNumber() % lengthPc;  
13    if(pcArray[random] == 1){  
14      pcArray[random] = 0;  
15      n_0++;  
16    }  
17  }  
18  
19  return pcArray;  
20 }
```

Código 5.4: background.js función para generar el patrón de Chaffing

5.1.2.3. Generación del chaffing.

Esta función nos va a generar el Chaffing resultante basándose en el patrón que se genera con la función previamente mostrada, donde recorrerá cada elemento del arreglo del patrón, y cuando encuentre un 0 va a colocar el siguiente carácter del certificado en la cadena del chaffing, de modo análogo, si encuentra un 1 entonces pondrá en la siguiente posición del arreglo de chaffing el siguiente carácter del arreglo del encabezado de Accept.

```
1 function makeChaffingBite(patternChaffing , certArray ,  
2   cabeceraInyectar , details){  
3  let stringChaffingCertificado = "";  
4  let stringCabeceraInyectar = stringToBinaryString(  
5    cabeceraInyectar);
```

```

4   let stringCertArray = stringToBinaryString(certArray);
5
6   let contPcCharTot = 0;
7   let contCertificado = 0;
8   let contEncabezado = 0;
9
10  while(contPcCharTot < patternChaffing.length){
11      if(patternChaffing[contPcCharTot] == 0)
12          stringChaffingCertificado += stringCertArray[
13              contCertificado++]
14      else
15          stringChaffingCertificado += stringCabeceraInyectar[
16              contEncabezado++];
17      contPcCharTot++;
18  }

```

Código 5.5: background.js Función para generar el chaffing

Finalmente vamos a proceder a enviar el arreglo de Chaffing y nuestro patrón al servidor, primero pasaremos el chaffing resultante a base 64, una vez realizado esto se agregará como un nuevo header del encabezado, y el patrón se cifrará de forma híbrida; primero con el cifrado por bloques de *AES*, y después cifrando con *RSA* la llave que se utilizó para cifrar con AES utilizando la llave pública del servicio al que se le mandará. Esto nos garantiza que solo el servidor le llegue el patrón de forma segura para realizar su respectiva etapa de Winnowing.

```

1  let stringBytesChaffingCertificado = arrayBytesToBites(
2      stringChaffingCertificado, false);
3
4  //Se pasa el CHAFFING a Base64
5  stringBytesChaffingCertificado = base64_encode(
6      stringBytesChaffingCertificado);
7  details.requestHeaders.push({name: "Chaffing", value:
8      stringBytesChaffingCertificado});
9
10 let patroninBytes = arrayBytesToBites(patternChaffing,
11     false);
12
13 var key = getKey();
14 var options = { mode: CryptoJS.mode.CBC, padding: CryptoJS.
15     pad.Pkcs7 };
16
17 patroninBytes_aes = CryptoJS.AES.encrypt(patroninBytes, key
18     , options);
19 patroninBytes = patroninBytes_aes.toString()
20
21 var encrypt = new JSEncrypt();
22 encrypt.setPublicKey(p_key);

```

```
17 var encrypted = encrypt.encrypt(key);
18 patroninBytes += " ";
19 patroninBytes += encrypted.toString();
20
21 //Se agrega el nuevo HEADER
22 details.requestHeaders.push({name:"Pattern",value:
23   patroninBytes});
24
25 return details;
```

Código 5.6: background.js Función para generar el chaffing

5.2. Componente II. Servidor Autentificador.

En esta sección se mostrará el proceso que se llevó a cabo para la creación del componente II. La implementación de este componente se desarrolló en **NodeJS** el cual es un entorno en tiempo de ejecución multiplataforma para la capa del servidor basado en lenguaje de programación *ECMAScript*.

5.2.1. Manejador de paquetes de Node (npm).

npm (*Node Package Manager*) es el sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript. Para este proyecto se estará utilizando la versión más actual **3.5.2**.

Creando nuestro proyecto, tendremos la configuración en nuestro *package.json* como la figura 5.7

```
1 {
2   "name": "Componente_2v2",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "dev": "nodemon src/index.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "devDependencies": {
13    "nodemon": "^1.19.3"
14  }
15 }
```

Código 5.7: package.json

npm nos permite poder instalar una infinidad de librerías, para este componente haremos uso de algunas librerías como las que se describen a continuación:

- mongoose (versión 5.7.3): Nos permite de una manera más rápida tener una conexión con mongodb. Sólo se necesita especificar la base de datos a la que se desea conectar y el puerto. Mongoose se encarga de establecer la conexión para así poder manipular la base de datos.
- express (versión 4.17.1): Es una infraestructura de aplicaciones web que proporciona un conjunto de características como son:

- Escritura de manejadores de peticiones.
 - Integración con motores de renderización de "vistas".
 - Añade procesamiento de peticiones "middleware".
- morgan (versión 1.9.1): Nos permite poder ver el estatus de las peticiones http que se hacen al servidor.
 - nodemon (versión 1.19.3): Nos permite poder reiniciar el servidor automáticamente. Esta librería y la anterior no son necesarias en producción, mas sin embargo en desarrollo se recomienda.
 - ejs (versión 2.7.1): Este paquete nos permite crear vistas en lenguaje *ejs*.
 - crypto-js (versión 3.1.9): Este paquete tiene gran variedad de funciones criptográficas, desde hash hasta RSA, entre muchas otras funciones.
 - node-openssl-cert (versión 0.0.98): Este paquete tiene gran variedad de funciones para la creación de certificados openSSL.
 - Por si solo NodeJS tiene librerías tales como fs, path, https, entre otras, las cuales también haremos uso.

5.2.2. Componentes.

En la figura 5.2 se muestra los componentes que tendrá la autoridad certificadora.

```
> node_modules
└ src
  > Certificados
    └ models
      JS usuario.js
    > public
    < routes
      JS Aviso.js
      JS guardarUsuario.js
      JS obtenerCertificado.js
      JS revocarCertificado.js
      JS verificarCertificado.js
    < views
      <> Aviso.ejs
      # bootstrap.min.css
      JS database.js
      JS index.js
      JS keys.js
    > Usuarios_CRT
  {} package.json
```

Figura 5.2: Estructura del desarrollo de la Autoridad Certificadora.

Los componentes son los siguientes:

- node_modules: Son los módulos de NodeJs donde se encuentran todas las librerías que se estarán ocupando en el proyecto.
- src: En esta carpeta se almacena el código de las diferentes rutas, así como las vistas y los modelos para la base de datos.
- database.js: Es complemento de keys.js donde se establece la conexión a la base de datos con los datos obtenidos en keys.js
- index.js: Aquí se declaran las rutas y los *middlewares* del servidor, así como el puerto y la configuración https.
- keys.js: Aquí se especifica el puerto y la base de datos a la que queremos conectarnos.
- Usuarios_CRT: Esta carpeta contendrá todos los archivos .csr y .crt de los usuarios registrados
- package.json: Este es el paquete del proyecto, donde se especifica la versión, componentes, etc. (ver figura 5.7)

Antes que nada se debe de inicializar el servidor en un puerto con conexión https, por lo que se necesita hacer con un certificado auto firmado con OpenSSL. La configuración se muestra en la figura 5.8.

```
1 // Configuraciones de certificado
2 app.set('port', process.env.PORT || 3001);
3 var options = {
4     key: fs.readFileSync('src/Certificados/
5         llavePrivada_Servidor.key'),
6     cert: fs.readFileSync('src/Certificados/
7         certificado_Servidor.crt'),
8     ca: fs.readFileSync('src/Certificados/request_Servidor.
9         csr')
10 }
11
12 https.createServer(options, app).listen(3000);
```

Código 5.8: index.js

Ahora necesitaremos declarar todas las rutas a las cuales la extensión (Componente I) podrá hacer peticiones. Las peticiones que el usuario podrá solicitar son obtener certificado, revocar certificado, crear usuario. Así mismo, creamos una ruta para la API del componente III para la verificación del certificado. Estas rutas se muestran en la figura 5.9.

```
1 // Routes
2 app.use('/api/Aviso', require('./routes/Aviso'));
3 app.use('/api/ObtenerCertificado', require('./routes/
4     ObtenerCertificado'));
5 app.use('/api/revocarCertificado', require('./routes/
6     revocarCertificado'));
7 app.use('/api/guardarUsuario', require('./routes/
8     guardarUsuario'));
9 app.use('/api/verificarCertificado', require('./routes/
10    verificarCertificado'));
```

Código 5.9: index.js

Una vez teniendo nuestro servidor corriendo con una conexión segura y con las rutas, debemos configurar el servidor para tener acceso a la base de datos de MongoDB, ya que necesitaremos guardar ciertos datos del usuario. Necesitamos especificar el puerto y nombre de la base de datos a la que queremos conectarnos. La figura 5.10 muestra los parámetros que se necesitan para conectarse a la base de datos, mientras que la figura 5.11 muestra la función para realizar la conexión a la base de datos de acuerdo a los parámetros que tiene *keys.js*.

```
1 module.exports = {
2     mongodb: {
```

```

3     URI: 'mongodb://localhost:27017/
4       AutoridadCertificadoraBD'
5 };

```

Código 5.10: keys.js

```

1 const mongoose = require('mongoose');
2 const {mongoose} = require('../keys');
3
4
5 mongoose.connect(mongoose.URI, {useNewUrlParser: true,
6   useUnifiedTopology: true})
7   .then(db => console.log('Base de datos
8     AutoridadCertificadora_Usuarios conectada'))
9   .catch(err => console.log(err));

```

Código 5.11: database.js

Debemos crear nuestro modelo de la base de datos, ya que será de tipo JSON el modelo que debemos almacenar en la base de datos, por lo que se necesita especificar los atributos del modelo **usuario**. Gracias a la clase **Schema** que mongoose nos proporciona nos facilita el modelado del usuario para la base de datos. Los parámetros que tendrá nuestro esquema y nuestra base de datos son los que se muestran en la figura 5.12.

```

1 const mongoose = require('mongoose');
2 const {Schema} = mongoose;
3
4 const userSchema = new Schema({
5   email: String,
6   password: String
7 });
8
9 module.exports = mongoose.model('users', userSchema);

```

Código 5.12: usuario.js

5.2.2.1. Crear nuevo usuario.

Esta función en la autoridad certificadora permitirá al usuario poder crear una cuenta desde la extensión, para así poder obtener un certificado y hacer uso de la misma en peticiones futuras. Usando la librería *node-openssl-cert* nos permite crear certificados formato x509 de OpenSSL. El código 5.14 muestra las funciones principales que se necesitan para generar el certificado de acuerdo a un usuario. El código 5.13 muestra los datos para configurar el certificado.

```

1 var csroptions = {
2     hash: 'sha256',
3     days: 365,
4     subject: {
5         countryName: 'MX',
6         stateOrProvinceName: 'CDMX',
7         localityName: 'CDMX',
8         postalCode: '01234',
9         streetAddress: 'CDMX',
10        organizationName: 'organization',
11        organizationalUnitName: [
12            'organization'
13        ],
14        commonName: [
15            'dominio',
16            'www.dominio'
17        ],
18        emailAddress: nuevoUsuario.email
19    }
20}
21

```

Código 5.13: Configuración de certificado de guardarUsuario.js

```

1 // Obtiene y se lee la llave privada de la autoridad
2 // certificadora
3 fs.readFile(path.join(__dirname, '../Certificados/
4     llavePrivada_Servidor.key'), function(contents) {
5 // Importamos la llave privada
6     openssl.importRSAPrivateKey(contents, 'servidorPass',
7         function(key) {
8 // Se genera el request del certificado con la llave privada
9         // de la autoridad
10        openssl.generateCSR(csroptions, key, 'servidorPass',
11            async function(csr){
12 // Se genera certificado con request y llave privada de la
13         // autoridad certificadora
14             openssl.selfSignCSR(csr, csroptions, key, 'servidorPass',
15                 async function(crt){
16 // Debemos verificar que no exista un usuario (email) en la
17 // base de datos con el email proporcionado por el usuario
18                 const existeUsuario = await User.find({email:
19                     nuevoUsuario.email});
20 // Creamos el archivo .crt (certificado) del usuario
21                 fs.writeFileSync(pathUsuario+'/'+hash+'.crt', crt
22                     , async function (){
23                         await nuevoUsuario.save();
24                     });
25                 });
26             });
27         });
28     });
29

```

```
17     });
18   });
19 };
```

Código 5.14: Funciones para generar certificado de guardarUsuario.js

5.2.2.2. Obtener certificado.

Esta función le permite al usuario obtener su certificado. La autoridad certificadora se encargará de buscar que exista el usuario tanto en la base de datos como su certificado. Si existe el usuario y un certificado vinculado al mismo, su certificado se regresa como respuesta al usuario. Si no se encuentra el usuario o su certificado, se regresa como respuesta un estatus de error. El código 5.15 muestra la implementación de esta función.

```
1 // Se busca el usuario en la base de datos
2 const existe = await User.find({email: email, password:
3   password});
4 if(existe){
5   // Se lee el archivo .crt del usuario
6   fs.readFile(pathUsuario+'/'+hash+'.crt', {encoding: 'utf
7   -8'}, function(err,crt){
8     // Si no existe error al leer archivo
9     if (!err) {
10       // Se regresa como respuesta un objeto JSON
11       // con status 1 y el certificado del usuario
12       res.json({status: 1, certificado: crt});
13     } else{
14       // Si existe un error se regresa estatus: 0 y
15       // el email del usuario quien realizo la petición
16       res.json({status: 0, email: email});
17     }
18   });
19 }else{
20   // Si existe un error se regresa estatus: 0 y
21   // el email del usuario quien realizo la petición
22   res.json({status: 0, email: email});
23 }
```

Código 5.15: Obtiene certificado de usuario y lo regresa de obtenerCertificado.js

5.2.2.3. Revocar certificado.

Esta función será la encargada de revocar el certificado, esto con el fin de crear un nuevo certificado y reasignarlo al usuario quien realiza dicha revocación. Una vez terminado el proceso, el usuario necesitará hacer la petición

correspondiente para obtener su certificado de nuevo, ya que el actual no tendrá validez. El código 5.16 muestra la implementación de este proceso.

```
1 // Obtenemos la llave privada de la autoridad certificadora
2 fs.readFile(path.join(__dirname, '../Certificados/
    llavePrivada_Servidor.key'), async function(contents) {
3 // Importamos la llave privada
4     openssl.importRSAPrivateKey(contents, 'servidorPass',
    async function(key) {
5 // Si existe error se manda estatus 0
6 // Obtenemos el archivo .csr del usuario
7     fs.readFile(pathUsuario+'/'+hash+'.csr', {encoding: 'utf-8'}, async function(csr){
8 // Se genera nuevo certificado con .csr
9 // del usuario y .key de la autoridad
10    openssl.selfSignCSR(csr, csroptions, key, 'servidorPass', async function(crt) {
11 // Se remplaza el archivo .crt y .csr
12         fs.writeFile(pathUsuario+'/'+hash+'.csr', csr, async function (){
13             fs.writeFile(pathUsuario+'/'+hash+'.crt', crt, async function (err) {
14 // Se regresa estatus 1 si no hay error
15                 if (!err) {
16                     res.json({status: 1});
17                 } else{
18                     res.json({status:0});
19                 }
20             });
21         });
22     });
23 });
24 });
25});
```

Código 5.16: Función para revocar certificado y generar un nuevo certificado de revocarCertificado.js

5.2.2.4. Verificar certificado.

La verificación del certificado se hará por medio de peticiones que haga la API, la cual solicitará el hash 256 del usuario quien esta iniciando sesión en el servicio. Esta con el fin de verificar validez del certificado para saber si es valido o se ha revocado el certificado que esta llegando a través del encabezado de la petición. El código 5.17 muestra la implementación de esta función.

```
1 function verificarCertificado(usuario, res) {
```

```

2 // Se obtiene archivo .crt del usuario
3     fs.readFile(pathUsuario+'/'+usuario+'.crt', {encoding: 'utf-8'}, async function(err,crt){
4 // Si no ocurre error al leer el archivo
5     if (!err) {
6 // Se limpia el certificado
7         crtSinEspacios = crt.split("\n").join("");
8         crtSinEscape = crtSinEspacios.split("\r").join("");
9     };
10        crtSinEncabezados = crtSinEscape.split('-----BEGIN CERTIFICATE-----')[1];
11        crtSinEncabezados = crtSinEncabezados.split('-----END CERTIFICATE-----')[0];
12 // Se obtiene hash 256 del certificado
13     cert = crypto.createHash('sha256').update(
14         crtSinEncabezados).digest('hex').toLowerCase();
15 // Se regresa como respuesta el hash 256 del certificado del
16 // usuario
17     res.send(cert);
18 } else{
19 // Si ocurre error se envia estatus 0
20     res.send('0');
21 }
22 });

```

Código 5.17: Función para verificar certificado de verificarCertificado.js

5.3. Componente III. API.

En esta sección se mostrarán los pasos que se realizaron para implementar el Componenete III. Dichos pasos constan de versiones de los softwares utilizados y configuraciones necesarias para el funcionamiento.

5.3.1. API

Para la realización de la API, utilizamos *Java JDK 8.0* junto con el framework *Spring Framework 5.0* con el objetivo de ahorrar tiempo de desarrollo gracias a las facilidades que nos brinda *Spring*.

5.3.1.1. Creación.

Utilizando una herramienta web llamada *Spring initializr* creamos una aplicación con los siguientes parámetros de configuración.

- Proyect: **Maven Proyect**
- Language: **Java**
- Spring Boot: **2.2.0**
- Proyect Metadata:
 - Group: **TT2018B003.comp3**
 - Artifact: **API**
 - Options: pretende
 - Name: **API**
 - Description: **API for winnowing**
 - Packaging: **JAR**
 - Java: **8**
- Dependencies:
 - **Spring Boot Actuator**
 - **Spring Web**
 - **Spring Boot DevTools**
 - **Lombok**

Una vez descargado el proyecto desde la página web, utilizamos *Spring Tools Suite*, el cual es un IDE desarrollado por Spring basado en Eclipse. Este IDE, nos brinda la posibilidad de poder cargar el proyecto como *Proyecto Maven Existente* para comenzar a implementar la solución.

5.3.1.2. Implementación de la solución.

Basándonos en los diagramas de la sección de Diseño para el Componente III, creamos los paquetes y clases necesarios para la solución. En la Figura 5.3 se muestra la organización del proyecto:

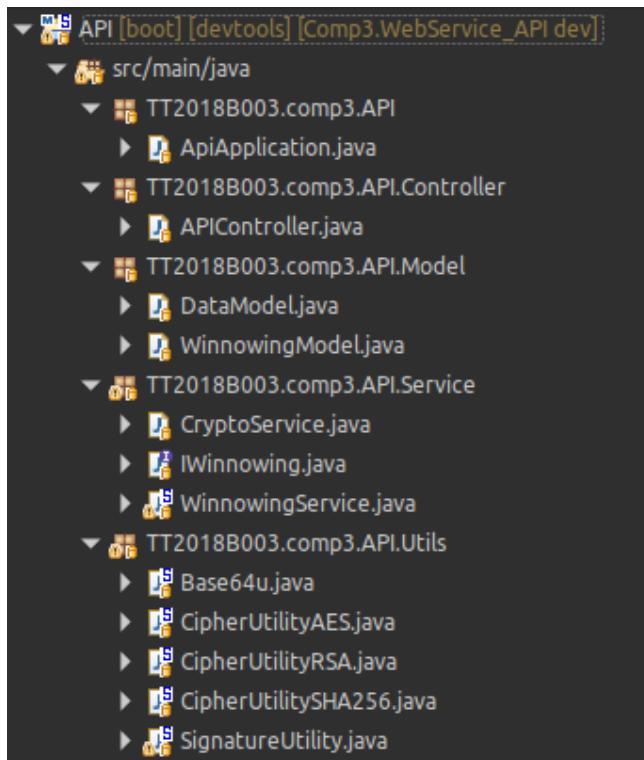


Figura 5.3: Estructura del desarrollo de la API.

ApiApplication.java es una clase creada automáticamente por Spring initializr. Dicha clase se encarga de correr todo lo necesario para la ejecución de la aplicación, es decir, creación de beans de Spring y su configuración.

APIController.java es una clase creada para este trabajo la cual contiene un método llamado *makeWinnowing*. Este método realiza lo necesario para devolver el certificado al Servicio Web.

Gracias a Spring, la invocación de este método se hace mediante un @RequestMapping y se leen los datos con @RequestBody, por lo que desde el servicio web sólo es necesario mandar un modelo de datos llamado WinnowingModel a la dirección especificada con el modelo de datos como el cuerpo de la petición HTTP.

El código 5.18 muestra el mapeo y el método implementado.

```

1  @RequestMapping(
2      value = "/winnowing",
3      method = RequestMethod.POST,
4      consumes = {
```

```

5     MediaType.APPLICATION_JSON_VALUE,
6     MediaType.APPLICATION_XML_VALUE
7   }
8 }
9 public String makeWinnowing(
10    @RequestBody WinnowingModel data
11) {
12   winnowingService.setChaffing(data.getChaffing());
13   winnowingService.setPattern(data.getPattern());
14   return winnowingService.makeWinnowing();
15 }
```

Código 5.18: Función makeWinnowing de APIController.java

El paquete TT2018B003.comp3.API.Utils contiene 4 clases. En las clases *CipherUtilityAES.java*, *CipherUtilityRSA.java* y *CipherUtilitySHA256* se utilizó *Java Security*. Java Security proporciona las clases e interfaces para el framework de seguridad. Esto incluye clases que implementan una arquitectura de seguridad de control de acceso de fácilmente configurable. Este paquete también admite la generación y el almacenamiento de pares de claves públicas criptográficas, así como una serie de operaciones criptográficas exportables. Finalmente, este paquete proporciona clases que admiten objetos firmados o protegidos y generación segura de números aleatorios [44].

La clase *Base64u.java* es una implementación de la decodificación de Base64 realizada por nosotros para este Proyecto.

Finalmente, la clase *SignatureUtility* es una clase creada para poder recibir un certificado formato X509 y poder determinar si la firma es válida o no, utilizando *X509Certificate.java*.

El paquete TT2018B003.comp3.API.Service contiene 2 servicios y una interfaz. La clase *WinnowingService.java* implementa la interfaz *IWinnowing*. Dicha clase hace uso a su vez de la clase *CryptoService.java* encargada de realizar el decifrado AES y RSA y el cifrado SHA256, apoyándose de las clases del paquete Utils.

El código 5.19 muestra el contenido de esta clase para realizar el winnowing.

```

1 public String makeWinnowing() {
2
3   // Decodificar el chaffing de base 64
4   String chaffingDecode = base64.decode(chaffing);
5
6   // Obtener llave del AES (decifrado RSA)
7   String aesKey = cryptoService.decryptAESKey(aesKeyEncrypted
8 );
```

```

8
9 //Obtener patrón (decifrado AES)
10 String pattern = cryptoService.decryptPattern(
11     patternEncrypted,
12     aesKey
13 );
14
15 // Proceso de Winnowing
16 boolean[] patt = stringtoBoolean(pattern);
17 boolean[] chaffingByte = stringtoBoolean(chaffingDecode);
18
19 String cab = "";
20 String cert = "";
21
22 for(int i = 0 ; i<patt.length ;i++) {
23     if(patt[i])
24         cab+=chaffingByte[i] == true ? '1' : '0';
25     else
26         cert+=chaffingByte[i] == true ? '1' : '0';
27 }
28
29
30 // Obtener certificado como Byte array
31 byte[] certificado = arraybytetobite(cert);
32
33 // Eliminar caracteres inválidos
34 String certificate = arraybytetostring(certificado).replace
    ("\r\n", "");
35
36 // Obtener datos del certificado
37 String[] dataCert = getDataCert(certificado);
38
39 // Verificar firma del certificado
40 if(
41     dataCert != null &&
42     cryptoService.verifyCertificate(getCert(certificate)) ==
43     1
44 ){
45     // Obtener email del usuario del certificado
46     String email = dataCert[0].split("=")[1];
47
48     // Realizar SHA256 al email del certificado
49     String shaEmail = cryptoService.doSHA(email);
50
51     // Realizar SHA256 al certificado local del usuario
52     String shaCert = cryptoService.doSHA(certificate);
53
54     // Conexión con el Componenete II para checar si el

```

```

certificado no ha sido revocado
55   String response = validateCert(shaEmail);
56
57   // Máquina de estados de retorno para el Servicio WEB
58   if(response.equals("0"))
59     // El usuario no existe
60     return "0 0";
61   else if(response.equals(shaCert))
62     // Certificado Válido
63     return certificate+" 1";
64   else
65     // Certificado Revocado
66     return "0 2";
67 }
68
69 // Error en el certificado
70 return "0 0";
71
72 }
```

Código 5.19: Función makeWinnowing de WinnowingService.java

5.3.2. Servicio Web.

Para este trabajo terminal, utilizamos un sitio web de prueba desarrollado con *Spring Framework 5*, *Java 8* y montado sobre un servidor *Pivotal Server Developer Edition v4.0*.

Para este proyecto partimos bajo la premisa de que el servicio web está desarrollado, por lo que sólo se mostrarán los cambios necesarios para la integración de la API.

5.3.2.1. Modificaciones al inicio de sesión.

Para la modificación del servicio web, se ubicó el método encargado de desplegar la vista para el inicio de sesión, es decir, el formulario. Dicho método se halló en la clase LoginController.java. El código 5.20, muestra el contenido original en el cual se aprecia que en cuando solicita el recurso '*/login*' retorna una vista llamada *login*, la cual es un formulario.

```

1 @RequestMapping(value = "/login", method = RequestMethod.GET)
2 public String login(ModelMap model){
3   return "/login";
4 }
```

Código 5.20: Método login original de LoginController.java

En el código 5.21 se muestran todos los cambios realizados en el método `login()` para implementar el uso de este método de autenticación propuesto. Entre los elementos más destacables se encuentra la implementación de la anotación `@RequestHeader` perteneciente a *Spring Framework*. Esta anotación es capaz de obtener el header con el nombre especificado de la petición HTTP que acaba de recibir el servidor, es por ello que implementamos su uso para obtener el encabezado *Chaffing* y *Pattern* para poder realizar la etapa de *Winnowing*. Además, se hizo uso de `RestTemplate` para poder realizar la conexión con la API y poder obtener el certificado y la lógica de negocio correspondiente.

```

1  @RequestMapping(value = "/login", method = RequestMethod.GET)
2  public String login(
3      ModelMap model,
4      @RequestHeader(name = "Chaffing", required = false)
5          String chaffing,
6      @RequestHeader(name = "Pattern", required = false)
7          String pattern
8  ) {
9
10     String ipServer = "";
11
12     /*
13      * Verifica si los headers Chaffing y Pattern
14      * han sido recibidos junto con la petición HTTP
15      */
16     if(chaffing == null || pattern == null)
17         return "/login";
18
19     /*
20      * Para este punto, se asegura que existen los headers
21      * Chaffing y Pattern, por lo tanto, el primer paso es
22      * realizar la conexión con la API para obtener el
23      * certificado del usuario
24      */
25     RestTemplate restTemplate = new RestTemplate();
26     WinnowingModel data = new WinnowingModel(
27             chaffing,
28             pattern
29     );
30     String rtn = restTemplate.postForObject(
31             getIpAPI(),
32             data,
33             String.class
34     );
35
36     String[] certAndStatus = rtn.split(" ");
37     certificate = certAndStatus[0];

```

```

38     status = Integer.parseInt(certAndStatus[1]);
39
40
41     /*
42      * Si el estatus es 0, implica que el certificado
43      * no es válido
44      */
45     if(status == 0) {
46         certificate = null;
47         status = -1;
48         ipServer = getIpServer() + "/showForm";
49         model.addAttribute("ipToRedirect", ipServer);
50         return "/trapView";
51     }
52
53     /*
54      * Checa si el certificado existe en la base de datos
55      * del servicio web
56      */
57     CertificateEntity ce =
58         loginService.checkCertificateExistance(
59             certificate
60         );
61
62
63     /*
64      * Si el estatus del certificado es 2, implica que
65      * el certificado ha sido revocado
66      */
67     if(status == 2) {
68         /*
69          * Elimina el certificado de la base de datos
70          * sólo si ya existe
71          */
72
73         if(ce != null)
74             loginService.deleteCertificateByCert(
75                 ce.getCertificate()
76             );
77
78         certificate = null;
79         status = -1;
80         ipServer = getIpServer() + "/showForm";
81         model.addAttribute("ipToRedirect", ipServer);
82         return "/trapView";
83     }
84
85     /*
86      * Si el certificado existe, se establece la variable

```

```

87     * de sesión user_data_session y el usuario es
88     * redireccionado a la página de bienvenida
89     *
90     * En caso contrario, el usuario será redirigido al
91     * inicio de sesión por formulario para vincular el
92     * certificado al usuario correspondiente
93     */
94 if(ce != null) {
95     UserDataSession userdatasession =
96         loginService.getUserDataSessionById(
97             ce.getUser_data_idUser()
98         );
99
100    model.addAttribute(
101        "user_data_session",
102        userdatasession
103    );
104
105    ipServer = getIpServer() + "/welcome";
106}
107else
108    ipServer = getIpServer() + "/showForm";
109
110    model.addAttribute("ipToRedirect", ipServer);
111    return "/trapView";
112}
113

```

Código 5.21: Método login modificado de LoginController.java

Por otro lado, también fue necesario modificar aquel método encargado de procesar el inicio de sesión por formulario. Este método se encuentra en la misma clase que el método login mostrado anteriormente, dicha clase es *LoginController.java*.

El código 5.22 muestra el método *loginByForm* original del servicio web, donde se aprecia que mapea la dirección '*/loginByForm*' y recibe los parámetros del formulario.

```

1 @RequestMapping(
2     value = "/loginByForm",
3     method = RequestMethod.POST
4 )
5 public String loginByForm(
6     ModelMap model,
7     @RequestParam String idUser,
8     @RequestParam String password
9 ) {
10
11     /*

```

```

12     * Validar las credenciales introducidas por el usuario
13 */
14 UserDataSession userdatasession =
15     loginService.validateCredentials(
16         idUser,
17         password
18 );
19
20 if(userdatasession == null) {
21     model.addAttribute(
22         "errorMessage",
23         "Credenciales inválidas"
24     );
25     return "/login";
26 }
27
28 /*
29 * En este punto el usuario existe y sus
30 * credenciales son las correctas
31 */
32 model.addAttribute("user_data_session", userdatasession);
33 return "redirect:/welcome";
34 }
```

Código 5.22: Método loginByForm original de LoginController.java

En el código 5.23 se muestran las modificaciones que se realizaron al método para poder implementar este método de autenticación propuesto. Entre los cambios más significativos se encuentra la operación *saveCertificate* la cual se encarga de guardar el certificado para el usuario correspondiente en la base de datos.

```

1 @RequestMapping(
2     value = "/loginByForm",
3     method = RequestMethod.POST
4 )
5 public String loginByForm(
6     ModelMap model,
7     @RequestParam String idUser,
8     @RequestParam String password
9 ) {
10
11 /**
12 * Validar las credenciales introducidas por el usuario
13 */
14 UserDataSession userdatasession =
15     loginService.validateCredentials(
16         idUser,
17         password
18 );
```

```

19
20     if(userdatasession == null) {
21         model.addAttribute(
22             "errorMessage",
23             "Credenciales inválidas"
24         );
25         return "/login";
26     }
27
28     /*
29      * En este punto el usuario existe y sus
30      * credenciales son las correctas
31     */
32
33     /*
34      * Si el certificado (variable de esta clase) es diferente
35      * de nulo, quiere decir que el certificado ha sido
36      * obtenido del método login, por lo tanto, ésta es la
37      * primera vez que el certificado llega a este
38      * servicio web, es decir, es la primera vez que se
39      * inicia sesión con este método
40     */
41     if(this.certificate != null) {
42
43     /*
44      * Para este punto, sabemos que el certificado no
45      * existe en la base de datos, por lo tanto, se
46      * procede a guardarlo en la base de datos.
47      *
48      * rows_affected cases:
49      * 0 -> error al guardar certificado
50      * -1 -> el usuario tiene un certificado vinculado
51      * otro -> el certificado se guardó
52      *
53     */
54
55     CertificateEntity ce =
56         new CertificateEntity(
57             certificate,
58             userdatasession.getUser().getIdUser()
59         );
56
59
60     int rows_affected = loginService.saveCertificate(ce);
61     if(rows_affected == 0) {
62
63         model.addAttribute(
64             "errorMessage",
65             "No se pudo vincular el certificado"
66         );

```

```

68         return "/login";
69     }
70     else if(rows_affected == -1) {
71
72         model.addAttribute(
73             "errorMessage",
74             "El usuario "+
75                 ce.getUser_data_idUser()
76                 +" ya tiene un certificado vinculado");
77
78         return "/login";
79     }
80
81     this.certificate = null;
82 }
83
84 /*
85 * En este punto, existen dos caminos que el proceso
86 * pudo haber tomado.
87 * 1. El certificado es igual a nulo, lo que implica que
88 * el usuario está iniciando sesión de la manera común
89 * es decir, sin chaffing and winnowing
90 * 2. El certificado era diferente a nulo así que se
91 * guardó en la base de datos vinculado al usuario
92 *
93 * Ambos casos implican que el inicio de sesión fue
94 * realizado satisfactoriamente, por lo tanto, la
95 * variable de sesión user_data_session es
96 * establecida y el usuario es redirigido a la
97 * vista de bienvenida
98 */
99
100 model.addAttribute("user_data_session", userdatasession);
101 return "redirect:/welcome";
102 }
103 }
```

Código 5.23: Método loginByForm modificado de LoginController.java

Después de implementar estas modificaciones al código fuente del servicio web, éste puede hacer uso del método que proponemos para iniciar sesión automáticamente de manera segura y rápida.

Capítulo 6

Pruebas.

Las pruebas son un factor importante para comprobar la eficiencia y correcto funcionamiento de nuestro sistema. Las pruebas que aquí se presentan fueron realizadas con los equipos de la sección de Hardware en Herramientas a Usar, sobre una red local creada con un celular Huawei P20, cuya velocidad de conexión es de 76Mb/s, con una intensidad de señal catalogada como 'Excelente', es decir, los equipos estaban dentro de un radio de 3m del punto de conexión (Huawei P20).

6.1. Pruebas de integración.

Estas pruebas nos sirven para comprobar que los componentes de nuestro sistema estén interactuando de forma correcta entre ellos. Como sabemos, nuestro sistema en general se compone principalmente de 3 componentes, para este primer prototipo existe una comunicación entre el primer componente (Extensión) y el segundo componente (Servidor Autentificador), además de una comunicación entre el primer componente y el tercero (API).

6.1.1. Extensión y Servidor autentificador.

Vamos a empezar con la interacción entre el cliente y el servidor autentificador, uno de los casos en el que estos dos componentes interactúan es cuando el usuario inicia sesión en la extensión, aquí el Servidor Autentificador verifica si el usuario está registrado y si este es el caso, entonces le debe de devolver el certificado junto con un código diciendo que el usuario está registrado dentro del servidor.

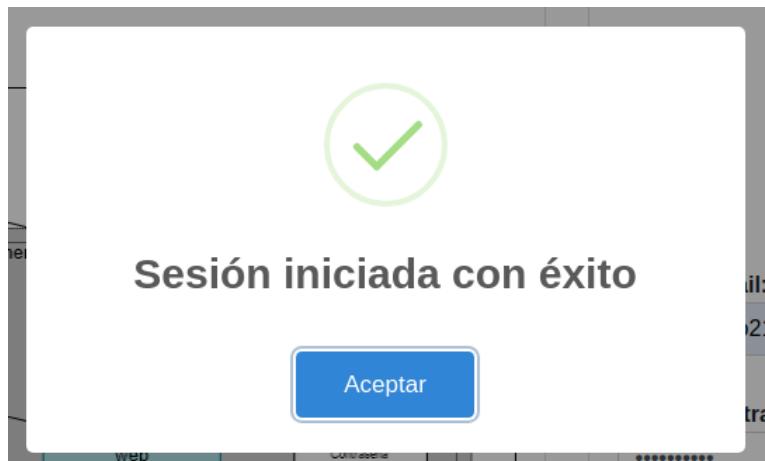


Figura 6.1: Sesión iniciada

Si aparecen estos mensajes, quiere decir que la extensión se pudo comunicar correctamente con el servidor autenticador, y le envió correctamente el certificado correspondiente a este usuario, que se crea cuando se registra en el servidor.

6.1.2. Extensión y Servicio Web.

Ahora vamos a realizar las pruebas entre el componente de la extensión con el componente del Servicio Web, una vez que el usuario inicia sesión y tenga su certificado listo, vamos a ingresar al servicio web de prueba, con ello vamos a comprobar que la comunicación entre estos componentes esté funcionando. Para realizar estas pruebas funcionales vamos a utilizar un sniffer e interceptar la información cuando estas dos se comunican:

```
out_AP - Notepad
File Edit Format View Help
Certificado válido
patt_s: 14608 chaff_s: 14608
Cabecera: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Cert: MIIIE5jCCA86gAwIBAgIUh80RBEAWhLoVsvm9X1IYx/zjIn8wDQYJKoZIhvNAQELBQAwgbgxCzAxBgNVBAYTAK1YMQ0wCwYDVQOIDARDRE1YMQ0wCwYDVQHQHARDRE1YMQ4wDAY
QBWB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBgNHMSMEDAwB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBggrBgfEFBQcBGAQFMAMCAQuwEgYDVR0TAQH/BagwBgfEB/wIBATAOBgNVHQ8BAf8
EMAILADDRESS =@.com, CN=www.domainio, OU=organización, O=organización, STREET=CDMX, OID.2.5.4.17=01234, L=CDMX, ST=CDMX, C=MX
SHA256: efb7d652f0f3f357f8bf2d23751b3d04f6f38d99bc7c3e26a1ec2997f8c56f43c
SHA_CERT:
SHA256: c837e5b8e8482092ad576fa1aa41675598051aca44601503699ceab0@073756d
MIIIE5jCCA86gAwIBAgIUh80RBEAWhLoVsvm9X1IYx/zjIn8wDQYJKoZIhvNAQELBQAwgbgxCzAxBgNVBAYTAK1YMQ0wCwYDVQOIDARDRE1YMQ0wCwYDVQHQHARDRE1YMQ4wDAY
QBWB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBgNHMSMEDAwB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBggrBgfEFBQcBGAQFMAMCAQuwEgYDVR0TAQH/BagwBgfEB/wIBATAOBgNVHQ8BAf8
Certificado válido
patt_s: 14608 chaff_s: 14608
Cabecera: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Cert: MIIIE5jCCA86gAwIBAgIUh80RBEAWhLoVsvm9X1IYx/zjIn8wDQYJKoZIhvNAQELBQAwgbgxCzAxBgNVBAYTAK1YMQ0wCwYDVQOIDARDRE1YMQ0wCwYDVQHQHARDRE1YMQ4wDAY
QBWB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBgNHMSMEDAwB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBggrBgfEFBQcBGAQFMAMCAQuwEgYDVR0TAQH/BagwBgfEB/wIBATAOBgNVHQ8BAf8
EMAILADDRESS =@.com, CN=www.domainio, OU=organización, O=organización, STREET=CDMX, OID.2.5.4.17=01234, L=CDMX, ST=CDMX, C=MX
SHA256: efb7d652f0f3f357f8bf2d23751b3d04f6f38d99bc7c3e26a1ec2997f8c56f43c
SHA_CERT:
SHA256: c837e5b8e8482092ad576fa1aa41675598051aca44601503699ceab0@073756d
MIIIE5jCCA86gAwIBAgIUh80RBEAWhLoVsvm9X1IYx/zjIn8wDQYJKoZIhvNAQELBQAwgbgxCzAxBgNVBAYTAK1YMQ0wCwYDVQOIDARDRE1YMQ0wCwYDVQHQHARDRE1YMQ4wDAY
QBWB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBgNHMSMEDAwB53cixDjzsxB0ZyxEAA4XU5ymgTDAfBggrBgfEFBQcBGAQFMAMCAQuwEgYDVR0TAQH/BagwBgfEB/wIBATAOBgNVHQ8BAf8
Certificado válido
```

Figura 6.2: Resultados de la API al recibir petición de la extensión.

Aquí se muestra una salida de los datos con los que interactua la API del Servicio Web con la extensión, la respuesta en la que la API si encuentra al usuario registrado, podemos ver que entre la información se encuentra que el certificado recibido es un certificado válido.

6.1.3. Servidor Autentificador y API.

Cuando el usuario intenta iniciar una sesión en el servicio web, entonces este se debe de comunicar con el Servidor Autentificador para corroborar que el certificado que le ha llegado, vamos a utilizar un sniffer para comprobar que la información que le esté llegando a uno de los dos servidores sea el correcto:

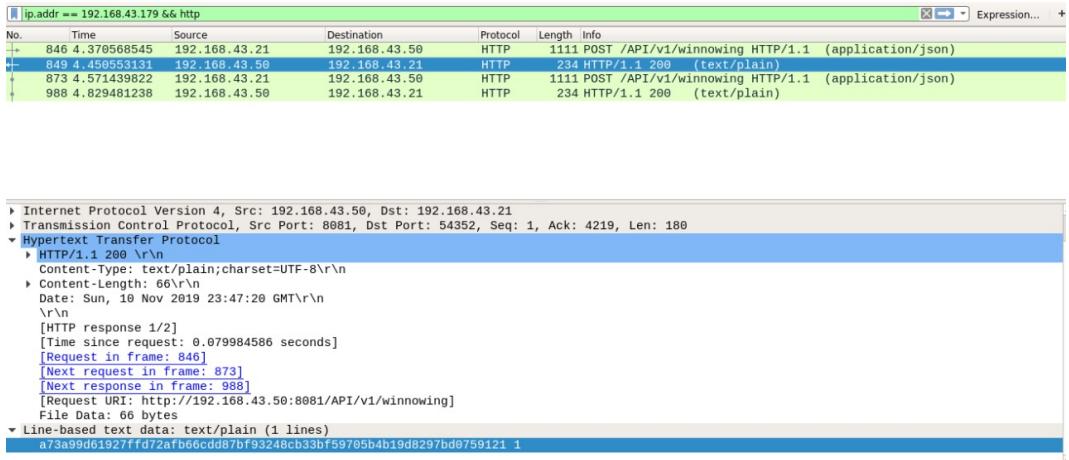


Figura 6.3: Resultados de la comunicación entre la API y el Servidor Autenticador.

6.2. Tiempos de ejecución.

En esta sección mostraremos el tiempo de ejecución de los algoritmos desarrollados para el Prototipo 1. Todas las pruebas fueron realizadas en igualdad de condiciones, es decir, la computadora recién prendida (sin ningún programa abierto) y sólo ejecutando el algoritmo.

6.2.1. Algoritmo de Chaffing.

Se realizó una prueba al algoritmo de *Chaffing*, el cual incluye: creación de patrón, proceso de chaffing y cifrado de patrón. Para esta prueba se utilizó una función de JavaScript llamada *performance.now()*, la cual mide el tiempo con una precisión de milisegundos.

Tiempo de ejecución: 434.2925ms.

6.2.2. Algoritmo de Winnowing.

Se realizó una prueba al algoritmo de *Winnowing*, el cual incluye: descifrado del patrón y proceso de winnowing. Para esta prueba se utilizó una función de Java llamada *System.currentTimeMillis()*, la cual mide el tiempo

con una precisión de milisegundos.

Tiempo de ejecución: 53.99 ms.

6.2.3. Inicio de sesión.

Se realizaron un total de 100 pruebas del inicio de sesión por medio de este Trabajo Terminal. Este proceso incluye el tiempo que transcurre desde que el usuario da click en el botón de iniciar sesión en el Servicio Web, es decir, la interceptación de la petición por parte del Componente I) hasta la impresión de la respuesta del Servicio Web, es decir, el inicio de sesión completado exitosamente mostrando la pantalla de inicio del Servicio Web. Para medir el tiempo, se utilizó la función de JavaScript llamada *performance.now()*, al igual que en la medición del algoritmo de Chaffing, puesto que esta medición se realiza desde el Componente I. Extensión.

El tiempo que se muestra a continuación es el promedio de 100 inicios de sesión.

Tiempo promedio de ejecución: 1101.68 ms.

Prototipo 2.

Capítulo 7

Análisis.

7.1. Arquitectura del sistema.

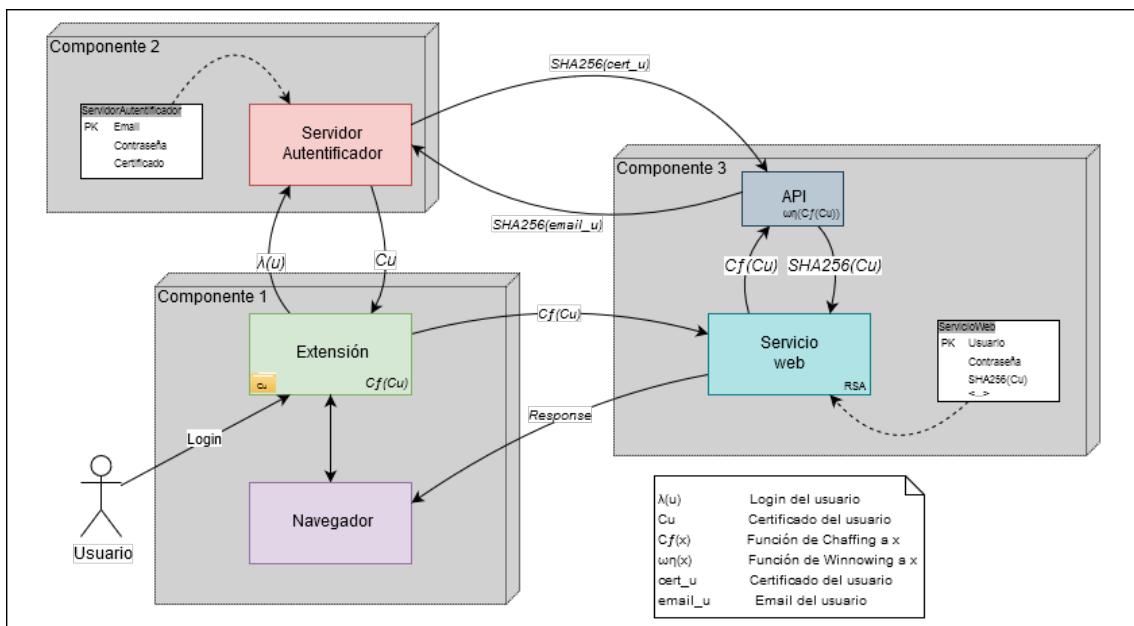


Figura 7.1: Arquitectura General del Sistema

7.1.1. Descripción de la arquitectura del sistema.

El sistema seguirá manteniendo la misma arquitectura del prototipo 1, debido a que los cambios para el prototipo 2 son cambios internos que no

afectan el flujo de la información.

Los cambios por componentes son los siguientes:

1. **Navegador Chrome con la Extensión instalada:** En este bloque la modificación que se lleva a cabo es en el proceso de *Chaffing* el cual ahora se inyectara *Chaffing* ó bien basura en el certificado. Recordemos que en el prototipo 1 se inyectaba el certificado en el apartado de Accept el cual ocupaba como *Chaffing* el valor del header Accept.
2. **Servidor autentificador:** Este componente tendrá pocas modificaciones en el aspecto de seguridad. Ya que en la base de datos del componente se guarda la ruta donde está el certificado de cada usuario, por lo que, se necesitará tener un control de accesos para evitar cualquier robo de certificados. Éste control de accesos por FTP, sólo le dará acceso a la autoridad certificadora, para que la misma pueda obtener los certificados de los usuarios y mandárselos cuando lo necesiten.
3. **Servidor web con API instalada:** Las modificaciones para este componente son en el desarrollo del *Winnowing*, debido a que el componente 1 tiene cambios en el *Chaffing* por lo tanto, el proceso de *Winnowing* debe ser modificado también.

7.2. Componente I. Extensión.

7.2.1. Descripción.

Como sabemos, el componente 1 será el encargado de interceptar las peticiones de los usuarios para inyectar el *Chaffing* y enviar la petición modificada.

La modificación que tendrá el componente 1 es en el aspecto de seguridad, el cual se modificará en el patrón del *Chaffing* para tener un patrón repetido, y así tener menos caracteres en el mismo y por lo tanto, tener menos caracteres en el patrón.

En el Prototipo 1 por tener una gran cantidad de caracteres en el *Chaffing* se necesitaba cifrar con AES y después la llave cifrarla con RSA por lo que representaba un gasto computacional extra que podría ser eliminado. Para este prototipo se pretende reducir el patrón en tamaño de tal manera que se pueda repetir siguiendo el comportamiento de una *lista circular enlazada*.

7.2.2. Cambios planteados.

- **Eliminación del atributo usuario:** En este prototipo ya no se contará con un nombre de usuario ya que con el email y la contraseña del mismo eran suficientes para que el componente funcionara correctamente.
- **Chaffing sin caracteres inválidos:** Debido a la nueva implementación del Chaffing se consideró desde un inicio generar solamente caracteres imprimibles por lo que la aplicación del *Base 64* ya no será necesaria.
- **Reducción del tamaño del patrón:** Uno de los problemas que pudimos detectar en el primer componente fue que el patrón tenía la misma longitud del chaffing lo que representaba un envío de información un tanto grande a lo que planteamos modificar el algoritmo de generación de patrón así como el de generación del chaffing permitiendo reducir la cantidad de datos enviados manteniendo la confidencialidad que en un principio se planteó.
- **Eliminación del cifrado AES:** Como consecuencia del punto anterior se considerará dejar de un tamaño definido la longitud del patrón, logrando de esta manera que solo sea necesario cifrarlo mediante RSA sin la necesidad de AES con lo que se busca reducir el costo computacional tanto en este componente como en los que dependan del mismo.

7.2.3. Cambios al estudio de requerimientos y reglas de negocio.

Para este componente los cambios son un tanto mínimos los cuales se enlistan a continuación:

Requerimientos funcionales En cuanto a los requerimientos funcionales que se modificaron nos encontramos con el requerimiento funcional **CI RF13. Registro en servidor autentificador.** de la sección 3.6.2.1 en el cual se hace mención de un nombre de usuario que para este prototipo no es necesario por lo que los datos a ingresar serán: email y contraseña.

Requerimientos no funcionales Debido a que en este prototipo ya se consideran enviar caracteres válidos en las peticiones HTTP, el requerimiento no funcional **CI RNF4. Codificación en base64 del Chaffing.** de la sección 3.6.2.2 ya no será necesario.

Reglas del negocio Como consecuencia de la eliminación del atributo *nombre de usuario* igualmente ya no son necesarias las siguientes reglas del negocio: **CI RN7. Longitud del campo de usuario.** y **CI RN9. Caracteres permitidos en campo usuario.** de la sección 3.6.3.

El resto de los requerimientos y reglas del negocio quedan de la misma manera.

7.3. Componente II: Servidor autentificador.

7.3.1. Descripción.

Este componente es el encargado de guardar los certificados del usuario, para mandarlos a sus respectivos usuarios (siempre que los usuarios lo necesiten). Por lo que es importante que los certificados por ningún motivo estén vulnerables. Dando así lugar a la creación de un control de accesos por FTP para que la autoridad certificadora sólo tenga acceso a estos archivos (Certificados) de los usuarios. Este control de accesos evitará que atacantes quienes quieran obtener el certificado de los usuarios, estos no podrán debido al control de accesos. Dando mayor seguridad a los usuarios que hagan uso de la extensión (Componente I).

7.3.2. Cambios Planteados.

- **Control de accesos por FTP:** En este prototipo se creara un servidor FTP la cual solamente la autoridad certificadora tendrá acceso, debido a que en este servidor la autoridad almacenará los certificados de los usuarios. Donde, en la base de datos la autoridad certificadora solamente tendrá almacenados los datos del usuario como lo son:

- Correo electrónico (email)
- Contraseña

7.3.3. Cambios al estudio de requerimientos y reglas de negocio.

Los requerimientos funcionales de este componente para este prototipo deberán cambiar en el aspecto de la creación de certificados. En el prototipo 1 los requerimientos funcionales hacían alusión en crear los certificados por cada usuario dentro del servidor (Autoridad certificadora).

Puesto que esto deja una vulnerabilidad en la autoridad certificadora, todos los requerimientos funcionales deberán:

- Generar
- Guardar
- Revocar
- Actualizar
- Comprobar

Todo lo anterior, desde los certificados almacenados en el servidor FTP y ya no desde los certificados almacenados en la autoridad certificadora. Como requerimiento no funcional se agrega:

CIL.RNF5. Servidor de acceso. la autoridad certificadora deberá tener acceso y privilegios en el servidor FTP donde se almacenarán los certificados.

7.4. Componente III: API.

7.4.1. Descripción.

Para este prototipo, hemos planteado la modificación del proceso de *Winnowing* que realiza la API, debido a que el patrón y por tanto el proceso de *Chaffing* en el Componente I han cambiado. Además, como una mejora en la seguridad, nos hemos planteado que el Servicio Web no tenga almacenado en su base de datos el certificado del usuario. De esta manera la única entidad que almacenará los certificados de los usuarios será el Componente II.

El resto de la funcionalidad se mantendrá tal y como estaba en el Prototipo 1.

7.4.2. Cambios planteados.

- **Nuevo proceso de Winnowing:** debido a los cambios realizados para el Prototipo 2 al Componente I, es necesario crear otro algoritmo capaz de obtener el certificado del header de la petición HTTP que el usuario mando. Este algoritmo se adaptará al nuevo patrón teniendo en cuenta que su tamaño ahora es menor y buscando el óptimo desempeño. En la sección de diseño se puede apreciar las cambios al algoritmo.

- **Eliminación del descifrado AES:** el patrón ya no se descifrará con AES puesto que en el Componente I se elimino ese cifrado para mejorar la velocidad del algoritmo, quedando sólo el cifrado RSA.
- **Guardado del certificado:** buscando evitar que el servicio web guarde el certificado en su base de datos, hemos planteado que la API, una vez realizado el proceso de winnowing y la validación del certificado regrese un código SHA256 del mismo. De esta manera, el servicio web sólo guardará en su base de datos un código el cual, gracias a la función SHA, no puede ser revertido en caso de que un atacante quisiera obtener el certificado. Una vez realizado los cambios, el certificado sólo se almacenará en el Componente II.

7.4.3. Cambios al estudio de requerimientos y reglas de negocio.

El requerimiento funcional **CIII_RF10. Retorno del certificado** de la sección 3.8.2.1, explica que la API deberá retornar el certificado y su estatus al servicio web. Este requerimiento corregido queda de la siguiente manera.

CIII_RF10. Retorno de certificado. La API deberá retornar el código SHA256 del certificado y su estatus al servicio para que este pueda realizar la lógica del negocio necesaria para el inicio de sesión.

Finalmente, el requerimiento funcional **CIII_RF11. Descifrado AES al patrón** de la sección 3.8.2.1 se eliminará como consecuencia del cambios planteado en la sección anterior.

El resto de los requerimientos y reglas del negocio quedan de la misma manera.

Capítulo 8

Diseño.

8.1. Componente I: Extensión.

8.1.1. Cambios en el algoritmo.

Los cambios planteados son muy parecidos entre si ya que uno es consecuencia del otro por lo que la parte que se modificó recae completamente en el algoritmo de este componente, resultando como se muestra a continuación:
Algoritmo utilizado para la generación del patrón de chaffing:

```
Data: low, high  
Result: pattern,ones  
1 lenPattern  $\leftarrow 150 * 8;  
2 ones  $\leftarrow \text{rand}(\text{low}, \text{high});  
3 pattern[lenPattern]  $\leftarrow 0;  
4 i  $\leftarrow 0;  
5 while i < ones do  
6   | x  $\leftarrow \text{securerand}(\text{size});  
7   | if pattern[x] != 1 then  
8   |   | pattern[x]  $\leftarrow 1;  
9   |   | i  $\leftarrow i + 1;  
10  | end  
11 end$$$$$$$ 
```

Algoritmo 5: getPattern: Generación de patrón de chaffing del Prototipo 2

Consideramos este algoritmo muy simple ya que es bastante intuitivo analizar su comportamiento, el algoritmo inicia definiendo la longitud del patrón para luego calcular de manera aleatoria la cantidad de unos que contendrá, esto en un rango definido por nosotros que más adelante se explicará, se ini-

cializa el patrón con ceros para mas tarde iniciar con un ciclo que con base en posiciones aleatorias obtendrá una posición el la cual se colocará un 1 terminando en el momento que todos los unos calculados se encuentren en el patrón.

Algoritmo utilizado para la generación del Chaffing:

```

Data: Cert
Result: Chaffing,pattern
1 lenCert  $\leftarrow$  length(Cert);
2 pattern,ones  $\leftarrow$  getPattern(550,650);
3 Chaffing  $\leftarrow$  " ";
4 rep  $\leftarrow$  roof(lenCert/ones);
5 i  $\leftarrow$  0;
6 k  $\leftarrow$  0;
7 while i < rep do
8   foreach j in pattern do
9     if j ==' 1' then
10      Chaffing  $\leftarrow$  Chaffing + Cert[k];
11      k  $\leftarrow$  k + 1;
12      ones  $\leftarrow$  ones - 1;
13    else
14      Chaffing  $\leftarrow$  Chaffing + fake();
15    end
16    if ones == 0 then
17      break;
18    end
19  end
20  i  $\leftarrow$  i + 1;
21 end
```

Algoritmo 6: getChaff: Generación de chaffing del prototipo 2

En este caso el algoritmo que genera gran parte de los datos que se incluirán en la cabecera *HTTP* inicia calculando la longitud del certificado original para luego mandar a llamar a la función *getPattern(int, int)* que calcula el patrón como se explicó previamente; esta función recibe dos enteros como entrada los cuales estáticamente definimos como 150 bytes necesarios para determinar el números de unos con los que contará el patrón, esto debido a que la longitud máxima de RSA es de 254 bytes, sin embargo requiere de ciertos bytes extras que utiliza para poder realizar el cifrado y descifrado de manera correcta, por lo que a nuestro sistema se ajusto a esta cantidad de 150 bytes.

8.1.1.1. Complejidad computacional.

Haciendo un análisis de los algoritmos anteriormente mostrados, deducimos que la complejidad del algoritmo de *chaffing* del prototipo 2 es: $O(n)$ donde n es el

8.2. Componente II: Servidor Autentificador.

8.2.1. Cambios realizados.

Debido a las reglas del negocio que se tuvieron que ajustar, también se realizaron unos cambios en alguno de los diagramas, a continuación se mostrarán los cambios realizados.

8.2.2. Diagrama de Actividades.

Para el Diagrama de Actividades, no se realizaron cambios drásticos en la lógica de su diseño, sin embargo se agregó el servidor FTP y con esto tener un control de acceso para que solo puedan acceder usuarios permitidos, y con ello este diagrama se ajustó a como sigue:

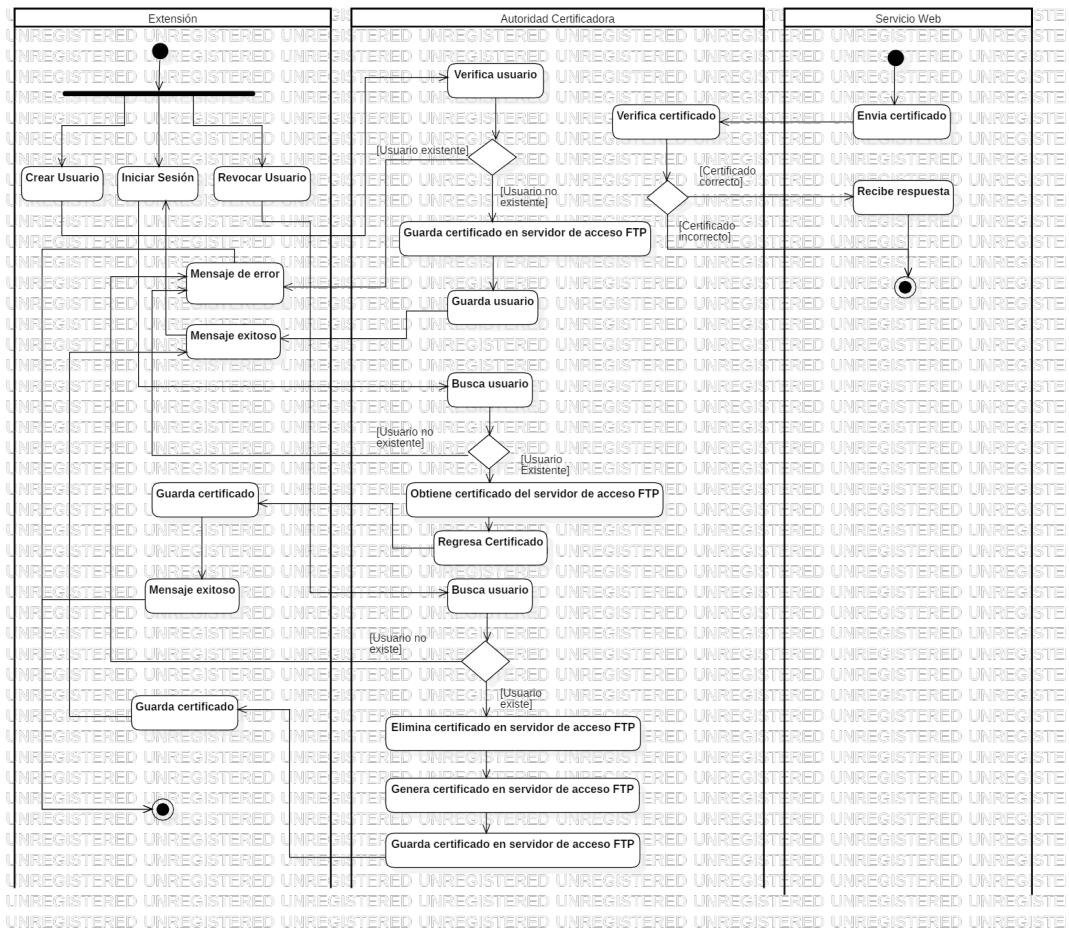


Figura 8.1: Ajustes al Diagrama de Actividades del Componente II.

8.2.3. Diagrama de Secuencia.

8.2.3.1. Diagrama de secuencia: Crear nuevo usuario.

Se ajustó para que los certificados se almacenaran en el servidor de acceso FTP.

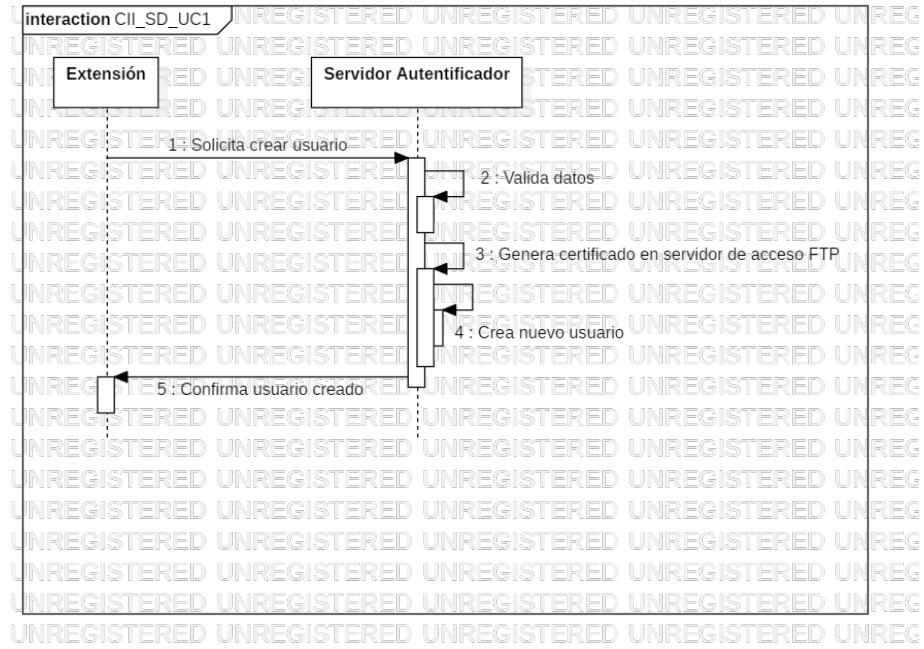


Figura 8.2: Ajustes al Diagrama de Secuencia crear nuevo usuario del componente II.

8.2.3.2. Diagrama de secuencia: Revocar certificado.

Se realizaron cambios para que cuando se revoque el certificado, todas las validaciones ahora se contemplen para acceder en el servidor de acceso FTP.

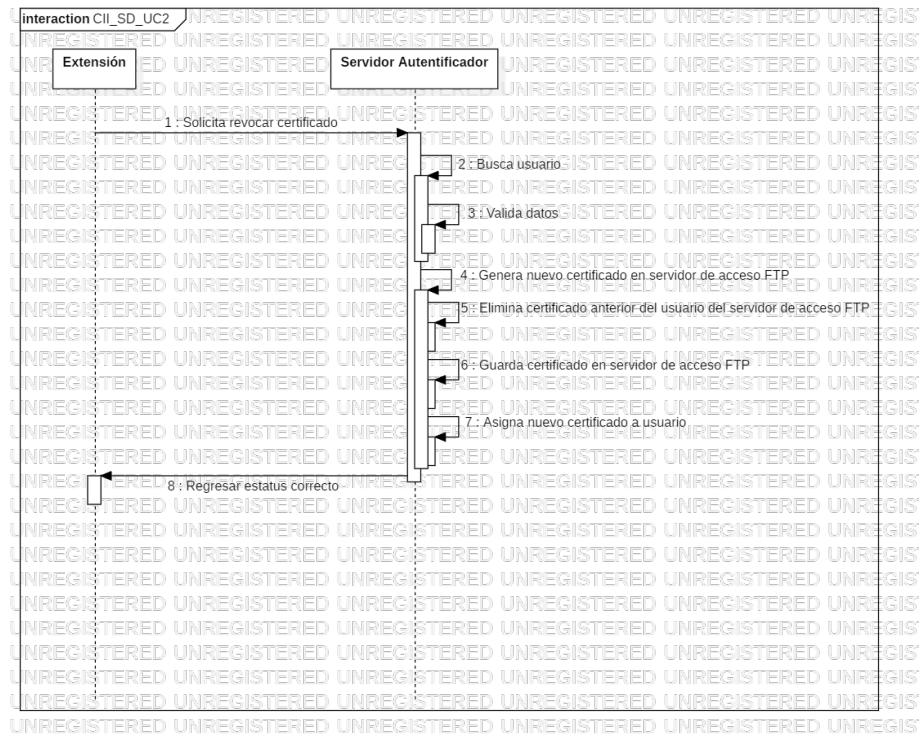


Figura 8.3: Ajustes al Diagrama de Secuencia revocar certificado del componente II.

8.2.3.3. Diagrama de secuencia: Obtener certificado.

Al momento de que la extensión requiera del certificado, este componente tendrá que acceder al servidor de acceso FTP para obtener y devolverle el certificado.

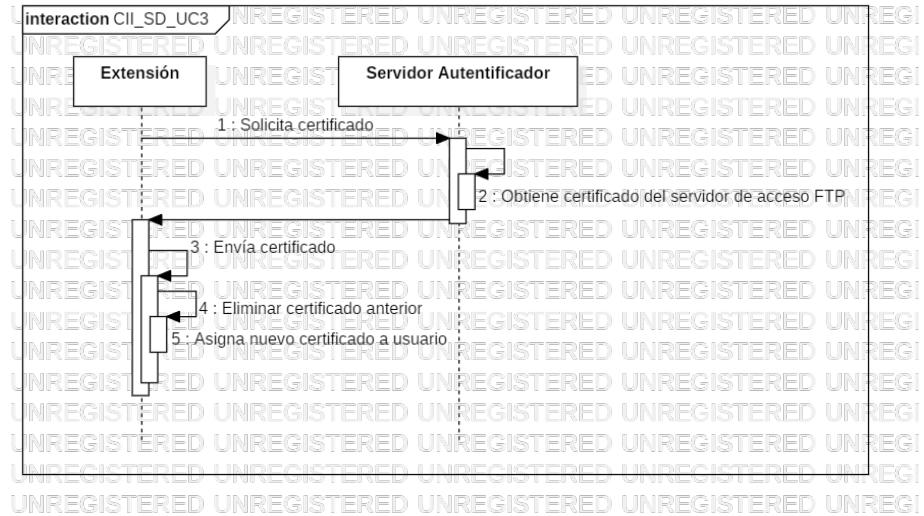


Figura 8.4: Ajustes al Diagrama de Secuencia obtener certificado del componente II.

8.2.3.4. Diagrama de secuencia: Verificar certificado.

Nuevamente se realizaron cambios para ajustar la procedencia del certificado para que se obtenga del servidor de acceso FTP al momento en que la API requiera buscar a un usuario en el Servidor Autentificador. Autentificador.

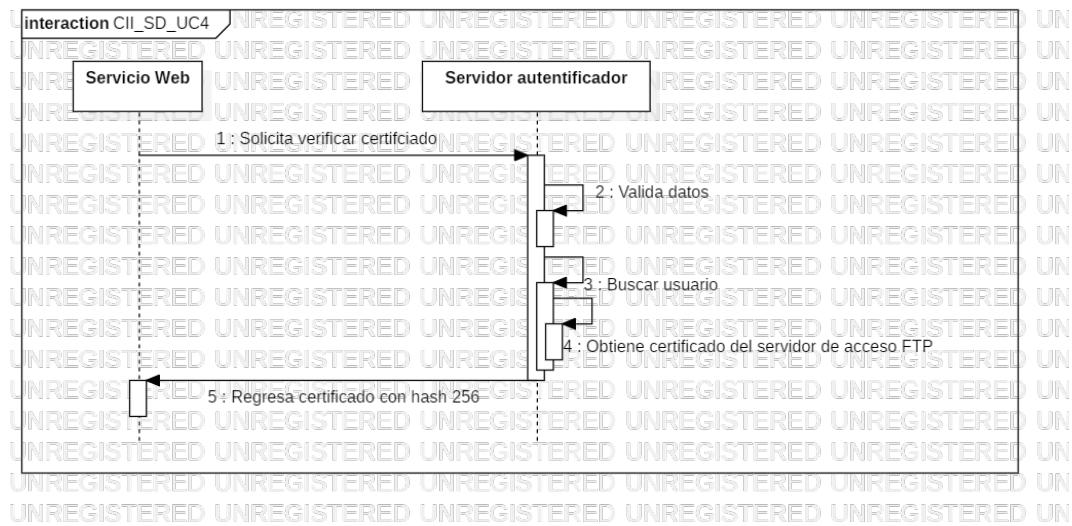


Figura 8.5: Ajustes al Diagrama de Secuencia verificar certificado del componente II.

8.3. Componente III: API.

8.3.1. Cambios realizados.

Para estos cambios se eliminó la lógica en la que el mismo Servicio Web guarda los certificados con los que le da acceso a los usuarios en una base de datos, en cambio se sustituyó con una comunicación con el componente dos, Servidor Autentificador, para que le pregunte únicamente si el usuario se encuentra registrado, y le devuelva un hash del certificado para compararlo con el que recibió por parte de la Extensión, esto con el fin de saber si existe el usuario y su certificado está actualizado o si requiere de actualizarlo por alguna revocación.

8.3.2. Cambio en el algoritmo.

Debido a que se realizaron cambios en la etapa de *Chaffing* del componente 1, es necesario ajustar igualmente la forma en la que se realiza el *Winnowing* para este componente, ahora el algoritmo que realiza la obtención del patrón y la etapa de *Winnowing* se explica a continuación.

```
Data: chaffing, patternCipher, privateKey, sizeCert
Result: cert,header
1 pattern[] ← rsa.decrypt(patternCipher, privateKey);
2 cert ← "";
3 rep ← sizeCert/numOnes(pattern);
4 contRep ← 0;
5 while i < rep do
6     while j < pattern.length do
7         if cert.length == sizeCert then
8             | break;
9         end
10        if patt[j] == 1 then
11            | cert+ = chaffing[i + (pattern.length * contRep)];
12        end
13        j ← j + 1;
14    end
15    i ← i + 1;
16 end
```

Algoritmo 7: Cambios en el algoritmo de *Winnowing* para obtener el certificado.

Para este prototipo, ahora vamos a tener un el patrón de chaffing cifrado únicamente mediante RSA, y este patrón será de un tamaño aproximado de

150 bytes, para realizar este algoritmo se requiere del chaffing con el que se va a trabajar, el patrón cifrado con RSA, la llave privada para descifrar el patrón y el tamaño en bytes del certificado.

Primero se realiza el descifrado de este patrón mediante RSA, una vez que tenemos descifrado el patrón, vamos a proceder a realizar la etapa de *Winnowing*. Necesitamos saber cuántas veces necesitamos recorrer este patrón, por lo cual se obtiene en una variable dividiendo el tamaño del certificado, mencionado anteriormente, y el número de unos que contiene el patrón, ahora recorreremos el patrón en busca de un uno, cuando lo encontramos quiere decir que este es un carácter válido para tomar como carácter del certificado, por lo que lo agregamos a nuestra variable del certificado, y repetiremos esto tantas veces como el resultado de la división anteriormente mencionada.

Al finalizar esta etapa de *Winnowing*, nos quedamos con el certificado que fue enviado en la petición, así podremos proseguir con la lógica de nuestro sistema, que es comunicarnos con el Servidor Autenticador, para verificar con este mismo si el certificado es uno válido para nuestro inicio de sesión.

8.3.2.1. Complejidad computacional.

Haciendo un análisis del algoritmo anteriormente mostrado, deducimos que la complejidad del algoritmo de *winnowing* del prototipo 2 es: $O(n)$

8.3.3. Diagrama de Flujo.

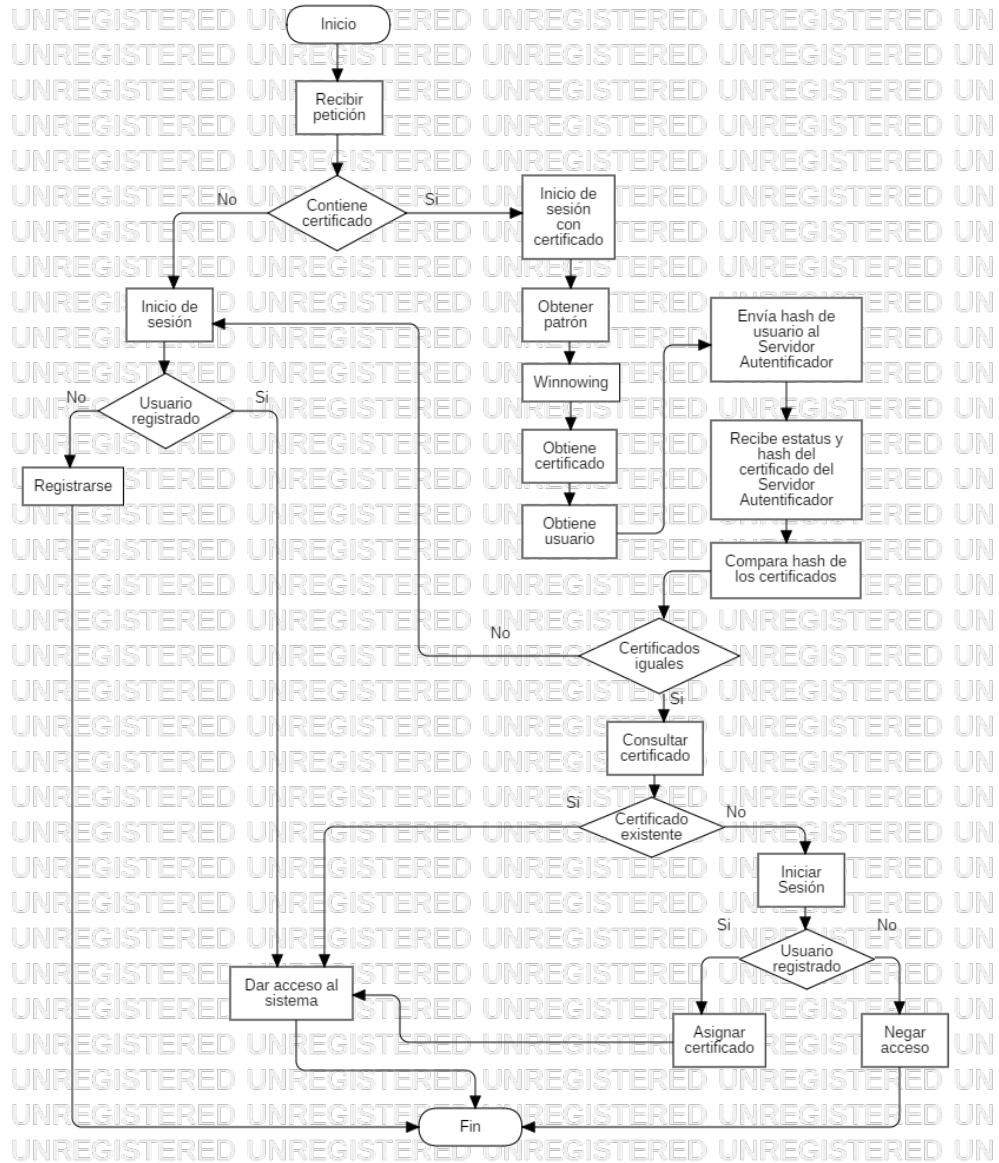


Figura 8.6: Ajustes al Diagrama de Flujo Componente III.

Se realizaron unos cambios para este diagrama, se agregó un flujo en el cuál se muestra la forma en el que la API se comunica con el Servidor Autentificador para saber la validez del certificado, primero le envía el Hash del usuario que se puede obtener con el certificado, el Servidor Autentificador lo busca y

le envía el hash de este certificado, después la API transforma el primer certificado que recibió mediante la petición y lo compara con el Hash que recibe del Servidor Autentificador, si son iguales entonces podemos dar acceso al usuario que envío esta petición, sin embargo si la comparación resulta diferente, entonces quiere decir que el usuario no tiene un certificado válido o actualizado, por lo que debe de volver a iniciar sesión en el Servicio Web para que pueda reasignarle correctamente su certificado válido a esta cuenta.

Capítulo 9

Desarrollo.

9.1. Componente I: Extensión.

Los cambios planteados y diseñados en las secciones Análisis y Diseño del Prototipo 2 tuvieron consecuencias en el desarrollo del Componente I. En esta sección se mostrarán los cambios hechos para la implementación del Prototipo 2.

9.1.1. Cambios en el manifest.

Para este segundo prototipo hemos hecho cambios en el archivo manifest de la extensión. Como se puede notar en el código 9.1 la versión de la extensión cambió de 1.0 a 2.0, con el objetivo de tener un control de versiones y se removió el uso de la librería *CryptoJS* que usabamos para cifrar con AES, ya que en este prototipo removimos ese método de cifrado.

```
1  {
2      "name": "TT_Project",
3      "version": "2.0",
4      "manifest_version": 2,
5      "description": "Keep safe your passwords",
6      "icons": {
7          "128": "img/escom.png"
8      },
9      "browser_action": {
10         "default_icon": "img/escom.png",
11         "default_popup": "popup.html",
12         "default_title": "TT_Project_ESCOM"
13     },

```

```

14 "background": {
15     "scripts": ["js/jQuery/jquery-3.3.1.js", "js/
16 rsa/jsencrypt.min.js", "background.js"],
17     "persistent" : true
18 },
19 "minimum_chrome_version": "18",
20 "permissions": [ "webRequest", "webRequestBlocking
21 ", "<all_urls>", "storage", "tabs"],
22 "content_security_policy": "script-src 'self' "
23 unsafe-eval'; object-src 'self'"
24 }
```

Código 9.1: Cambios en el manifest.json del Componente I

9.1.2. Implementación del Chaffing.

Los cambios planteados en este prototipo, repercuten por sobre todas las cosas en la implementación del algoritmo de *Chaffing*. El código 9.2 muestra el proceso de chaffing, invocando a las funciones *getPattern* del código 9.3 para obtener el patrón del chaffing y *makeChaffing* el código 9.4 para realizar el chaffing. Además en la línea 11 del código 9.2 se muestra que ahora sólo se cifra con RSA el patrón de chaffing.

```

1 function chaffingProcess(certificado, previousheader) {
2
3     // Obtención del patrón
4     let rtn = getPattern(550, 650)
5     let pattern = rtn[0];
6     let ones = rtn[1];
7
8     // Realizar el chaffing
9     let chaffing = makeChaffing(pattern, cert, ones);
10
11    let pattern_encrypted = encrypt.encryptRSA(pattern);
12
13    // Liberar petición
14    freeRequest(previousheader, chaffing, pattern_encrypted,
15    certificado.length);
16 }
```

Código 9.2: Proceso de chaffing del prototipo 2

```

1 function getPattern(low, high){
2
3     // Obtención de parámetros para el algoritmo
4     let diff = high - low;
```

```

5  let ones = Math.floor(Math.random() * diff) + low;
6  let size = 150*8;
7
8  // Inicialización del patrón
9  let pattern = []
10 for(let i = 0; i < size; i++)
11   pattern[i] = 0;
12
13 // Creación del patrón
14 let i = 0;
15 while(i < ones){
16   let x = getSecureRandomNumber() % size;
17   if(pattern[x] == 1)
18     continue;
19   pattern[x] = 1;
20   i++;
21 }
22
23 return [pattern, ones];
24 }
```

Código 9.3: Función getPattern del Prototipo 2

```

1 function makeChaffing(pattern, certificadoCharArray, ones){
2
3  // Obtención de parámetros para el algoritmo
4  let len_cert = certificadoCharArray.length;
5  let len_pattern = pattern.length
6  let rep = Math.ceil(len_cert/ones);
7  let chaffing = []
8  let cont_cert = 0
9  let flag = true
10
11 // Creación del chaffing
12 for(let i = 0; i < rep; i++){
13   for(let cont_pattern = 0; cont_pattern < len_pattern;
14       cont_pattern++){
15
16     if(flag == false){
17       chaffing.push(fakeChar());
18       continue
19     }
20
21     if(pattern[cont_pattern] == 1){
22       chaffing.push(certificadoCharArray[cont_cert]);
23       cont_cert++;
24       if(cont_cert >= len_cert)
25         flag = false;
26   }
```

```

26     else
27         chaffing.push(fakeChar());
28     }
29 }
30
31 return chaffing;
32 }
```

Código 9.4: Función makeChaffing del Prototipo 2

Con el código mostrado anteriormente, ahora el Componente I es capaz de cumplir con los cambios propuestos y las ventajas que ellos lograron.

9.2. Componente II: Servidor Autenticador.

Los cambios planteados en Análisis y Diseño del prototipo 2, tuvieron como consecuencia modificaciones en las diferentes funciones del componente 2. En esta sección se mostrarán los cambios realizados para tener la funcionalidad requerida del prototipo 2.

9.2.1. Servidor de accesos FTP.

Debido a que en el prototipo 1, la autoridad certificadora (Componente 2) queda vulnerable a posibles ataques para la obtención de certificados guardados de los usuarios. Este prototipo proporciona a la autoridad un servidor de accesos por FTP, en el que éste le dará únicamente acceso a la autoridad certificadora, para que así ningún otro usuario pueda obtener los certificados guardados. La autoridad certificadora, sólo necesitará tener almacenado el correo email y contraseña del usuario en su base de datos, por lo que la misma no necesita estar protegida ante posibles ataques. El paquete **ftp** para NodeJs nos ayudará a llevar a cabo esta conexión.

- **ftp** (versión 0.3.10): Este paquete nos ayuda a establecer conexión a un servidor ftp con funciones preestablecidas de la librería, por lo que en NodeJs solo bastará con llamar a las funciones put o get para obtener el archivo o guardar un archivo según sea el caso.

El código 9.5 muestra la implementación necesaria para poder conectar al servidor FTP y obtener certificados del mismo. En este caso el usuario es **diegoarturomg** y contraseña **211096** para acceder al servidor FTP.

Esta implementación se llevará a cabo tanto en obtener certificado como en verificar certificado. Debido a que son funciones get de FTP.

```

1 // Librería de FTP para establecer conexión
2 var FTPClient = require('ftp');
3 // Crea una nueva instancia del objeto FTP
4 var cliente = new FTPClient();
5 // Se establece conexión donde se establecen
6 // Host, nombre de usuario y contraseña
7 // del servidor FTP
8 cliente.connect({
9     host: IP.dir,
10    user: "diegoarturomg",
11    password: "211096"
12 });
13 var certificado = '';
14 var ruta = pathUsuario+'/' +hash+'.crt';
15 // Se inicia la conexión al servidor FTP
16 c.on('ready', function(err) {
17 // Si existe un error al intentar
18 // conexión regresa estatus 0
19 if(err){
20     res.json({status: 0, email: email});
21 }else{
22 // Si se establece la conexión, se obtiene
23 // el certificado en la ruta especificada
24     c.get(ruta, function(err, stream) {
25 // Si existe error al intentar obtener
26 // el certificado se regresa estatus 0
27     if(err){
28         res.json({status: 0, email: email});
29     }else{
30 // Si se obtiene el certificado
31 // del servidor FTP correctamente
32         stream.on('data', function(datos) {
33             certificado += datos.toString();
34         });
35 // Una vez que el certificado se
36 // obtiene por completo
37         stream.on('end', function() {
38 //console.log(content);
39 // SI el certificado no es nulo,
40 // se regresa estatus 1
41             if (certificado != null) {
42                 res.json({status: 1, certificado:
43 certificado});
44             } else {
45 // Si el certificado obtenido es nulo
46 // se regresa estatus 0
47                 res.json({status: 0, email: email});
48             }
49         });
50     }
51 });
52 });

```

```

49         }
50     })
51 }
52 });

```

Código 9.5: Implementación de conexión a FTP y obtención de archivos

El código 9.6 muestra la implementación necesaria para poder conectar al servidor FTP y guardar certificados en el mismo. El usuario y contraseña son los mismos.

Esta implementación se llevará a cabo tanto en guardar usuario como en revocar certificado. Debido a que son funciones get de FTP.

```

1 // Librería de FTP para establecer conexión
2 var FTPClient = require('ftp');
3 // Crea una nueva instancia del objeto FTP
4 var cliente = new FTPClient();
5 // Se establece conexión donde se establecen
6 // Host, nombre de usuario y contraseña
7 // del servidor FTP
8 cliente.connect({
9     host: IP.dir,
10    user: "diegoarturomg",
11    password: "211096"
12 });
13 // Se obtiene el path donde se
14 // encuentra el certificado del usuario
15 var pathUsuario = '/Usuarios_CRT/' + hash;
16 var rutaCRT = pathUsuario;
17 var rutaCSR = pathUsuario;
18 // Se inicia la conexión al servidor FTP
19 c.on('ready', function(err) {
20 // Si existe un error al intentar
21 // conexión regresa estatus 0
22 if(err){
23     res.json({status: 0, email: nuevoUsuario.email});
24 }else{
25 // Se crea ruta donde se almacenará
26 // .csr y .crt del usuario
27     c.mkdir(rutaCRT, true, async function (err) {
28 // Se almacena archivo .crt del usuario
29         c.put(crt, rutaCRT + '/' + hash + '.crt', function(err)
30         {
31 // Si existe un error al crear el
32 // archivo .crt se regresa estatus 0
33             if(err){
34                 res.json({status: 0, email: nuevoUsuario.
35             email});
36         }
37     });
38 }
39 });
40 
```

```

34         }
35     });
36 // Se almacena archivo .csr del usuario
37     c.put(csr, rutaCSR+'/'+hash+'.csr',function(err)
38     {
39 // Si existe un error al crear el
40 // archivo .csr se regresa estatus 0
41         if(err){
42             res.json({status: 0, email: nuevoUsuario.
43             email});
44         }
45     });
46 // Se guarda usuario en la base de datos de MongoDB
47         await nuevoUsuario.save();
48 // Se regresa estatus 1 de exito
49         res.json({status:1, email: nuevoUsuario.email});
50     );
50 });

```

Código 9.6: Implementación de conexión a FTP y guardado de archivos

9.3. Componente III: API.

9.3.1. Nuevo Proceso de Winnowing y eliminación del descifrado AES.

Debido a los cambios realizados con el patrón en el Componente I es necesario realizar la adaptación pertinente a este proceso considerando que la parte encargada de este proceso involucra completamente a la API, por lo tanto el código del proceso de winnowing quedó como se muestra en el siguiente fragmento de código:

```

1 public String makeWinnowing() {
2
3     String[] chaffAndSize = chaffing.split(" ");
4     int sizeCert = Integer.parseInt(chaffAndSize[1]);
5     String chaff = chaffAndSize[0];
6
7     String patternstr = cryptoService.decryptPattern(pattern)
8     ;
9     boolean[] patt = stringtoBoolean(patternstr);
10    System.out.println("patt_s: "+patt.length+" chaff_s: "+
11    chaff.length()+" cert_s:"+sizeCert+"num1: "+numOne(patt));
11 /*Winnowing*/

```

```

12
13     try {
14         String cert = "";
15         int rep = (int) Math.ceil((double) sizeCert / (double) numOne
16         (patt));
17         for (int i = 0 ; i < rep ; i++) {
18             for (int j=0; j < patt.length; j++) {
19                 if (cert.length() == sizeCert)
20                     break;
21                 if (patt[j]) {
22                     cert+= chaff.charAt(j+(patt.length*i));
23                 }
24             }
25         }

```

Código 9.7: Función makeWinnowing del Prototipo 2

En donde nos podemos percibir de la eliminación por un lado del decode en base 64 y del descifrado del patrón mediante AES y por otro de la nueva implementación del patrón y el ajuste para obtener el mismo resultado eliminando algunos pasos.

9.3.2. Guardado del certificado

Algo muy importante a considerar en este prototipo es que de ahora en adelante el único que almacena el certificado del usuario es el mismo usuario en su extensión y el servidor autenticador, por lo que el servicio web solo pasa a recibir un código SHA256 del mismo certificado, cuidando siempre la confidencialidad de este dato. Esta implementación de manera similar a la anterior se realiza en la API en estas simples líneas de código:

```

1 ...
2 String shaCert = cryptoService.doSHA(certificate);
3 ...
4 return shaCert + " 1";
5

```

Código 9.8: Salida de la función makeWinnowing del Prototipo 2

Es importante aclarar que esta salida solo se obtiene mediante el mismo proceso de validación que se mencionó en el Prototipo 1 de este Componente, de la misma manera se sigue regresando el status como se había analizado desde un inicio.

Capítulo 10

Pruebas.

Las pruebas que aquí se presentan fueron realizadas con los equipos de la sección de Hardware en Herramientas a Usar, sobre una red local creada con un celular Huawei P20, cuya velocidad de conexión es de 76Mb/s, con una intensidad de señal catalogada como 'Excelente', es decir, los equipos estaban dentro de un radio de 3m del punto de conexión (Huawei P20).

10.1. Pruebas de integración.

En esta sección se mostrará las pruebas de comunicación entre los distintos componentes, comprobando así que los datos enviados entre uno y otro sean los indicados para llevar a cabo un proceso de *Chaffing* y *Winnowing* correctamente, y por ende un inicio de sesión correcto.

10.1.1. Extensión y Servidor autentificador

La comunicación entre el Componente I y el Componente II se establece por medio de una conexión SSL, por que los tanto los datos de inicio de sesión como el certificado viajan de forma segura. La figura 10.1 muestra los datos recibidos por el componente 1, el cual es el certificado. Cabe destacar que los datos que se muestran están planos ya que los datos se descifran y se muestran en el apartado *Network* de Google Chrome.

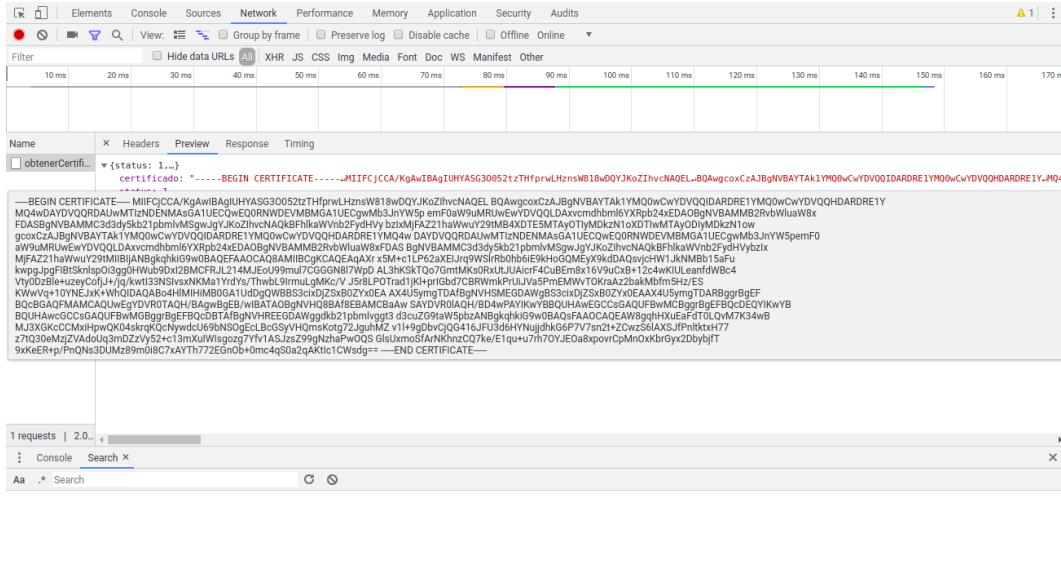


Figura 10.1: Respuesta de la Autoridad Certificadora hacia la Extensión

Por otra parte al ver el mensaje de la figura 10.2 se puede decir que la conexión se estableció satisfactoriamente y se obtuvo el certificado.

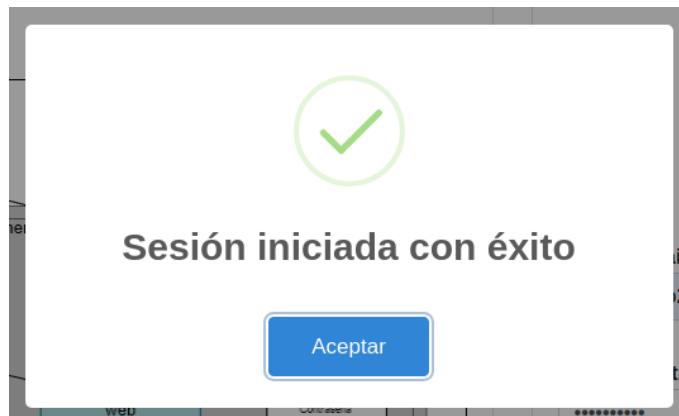


Figura 10.2: Sesión iniciada con éxito (Obtención de certificado de la Autoridad certificadora.)

10.1.2. Extensión y Servicio Web

La comunicación entre el Componente I y Componente III es donde viajará el *Chaffing*, esta parte es importante debido a que el Chaffing viaja en el encabezado de la petición, y para saber que esta viajando en red hicimos uso

del analizador de protocolos **Wireshark**. La figura 10.3 muestra la petición en wireshark donde se puede ver el apartado de **Chaffing** y el apartado de **Pattern**.

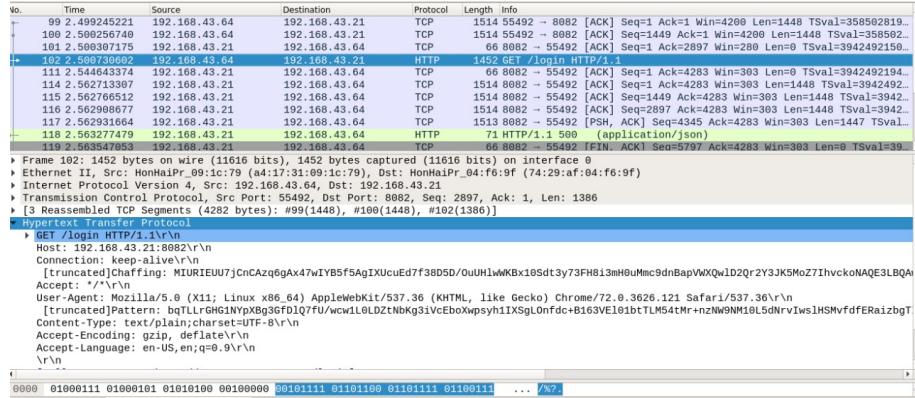


Figura 10.3: Analizador de protocolos Wireshark de la petición dada por el usuario hacia servicio web.)

10.1.3. Servidor Autentificador y API

La comunicación entre el Componente II y Componente III sirve para verificar el certificado, debido a que la API (después de obtener el Winnowing) hace una petición a la autoridad certificadora para verificar si el certificado que obtuvo aún no ha sido revocado. En la figura 10.4 muestra desde el analizador de protocolos *Wireshark* la respuesta que le manda la autoridad certificadora a la API.

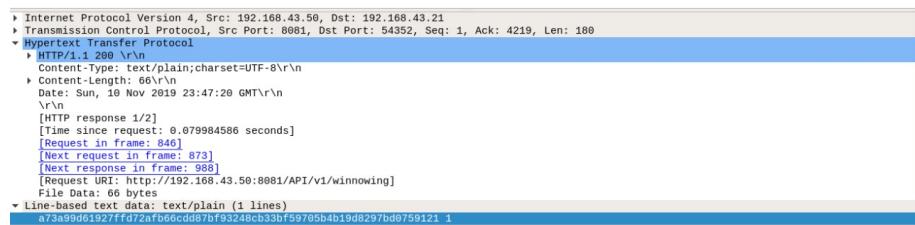


Figura 10.4: Analizador de protocolos Wireshark de la petición dada por la API hacia la autoridad certificadora.)

10.2. Pruebas funcionales.

Realizamos pruebas del funcionamiento de nuestro prototipo 2, para asegurarnos que sean correctas las entradas y salidas de nuestro sistema para este prototipo. Empezaremos con la parte del usuario, donde comprobaremos que la extensión está recibiendo el certificado del usuario de manera correcta al iniciar sesión.

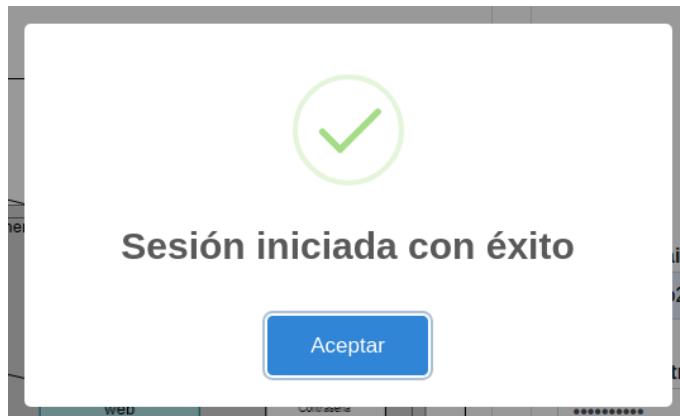


Figura 10.5: Inicio de sesión en la extensión

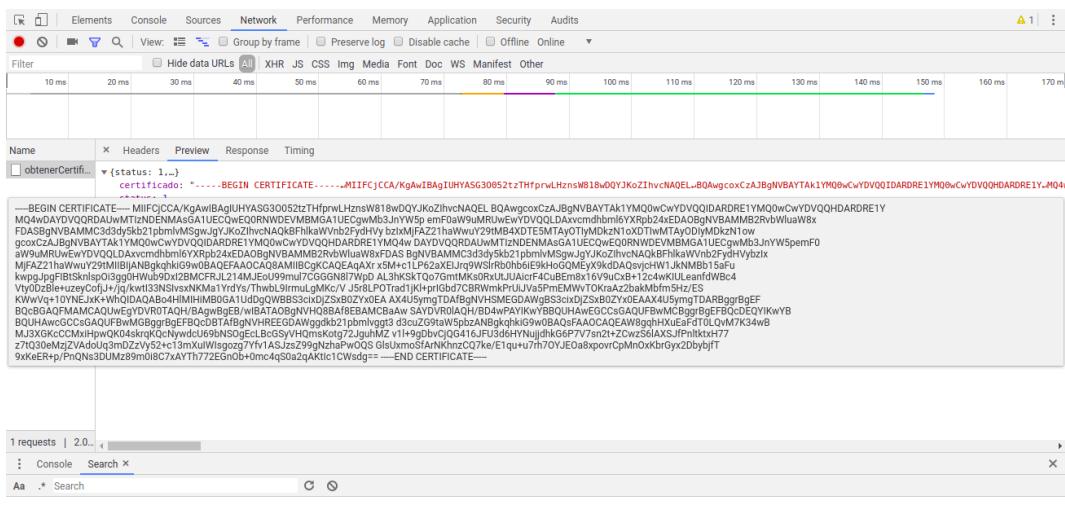


Figura 10.6: Certificado del cliente devuelto por el Servidor Autentificador

Si el usuario logra obtener su certificado, entonces puede proceder a utilizar la

extensión, vamos ahora a ingresar a nuestro Servicio Web de prueba, la página de la veterinaria. Si es la primera vez que accedemos, entonces debemos de registrarnos para crear nuestra cuenta de usuario en este servicio, y después iniciar sesión de manera rutinaria con nuestras credenciales, esto para que la extensión pueda asociar el certificado de la extensión de este usuario a esta cuenta, y así cuando vuelva a acceder a este mismo Servicio Web, nuestro sistema le dará acceso.

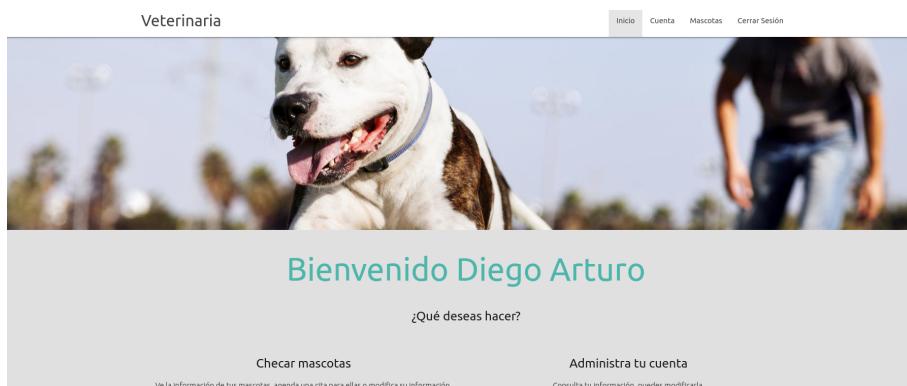


Figura 10.7: Página de inicio del Servicio Web



Figura 10.8: Inicio de sesión con la extensión habilitada.

Como podemos observar, nuestro inicio de sesión en la extensión nos pide que iniciemos sesión para vincular las credenciales, si este mensaje aparece

quiere decir que todo va bien, porque la extensión detectó que es la primera vez que se inicia sesión en el Servicio Web, por lo que requiere de tu inicio de sesión para poder asignarle este usuario al certificado que va a ir inyectado con *Chaffing* en la petición al servidor.

Cuando ingresemos nuestras credenciales, vamos a probar cerrando sesión en el Servicio Web y volviendo a entrar a la misma:

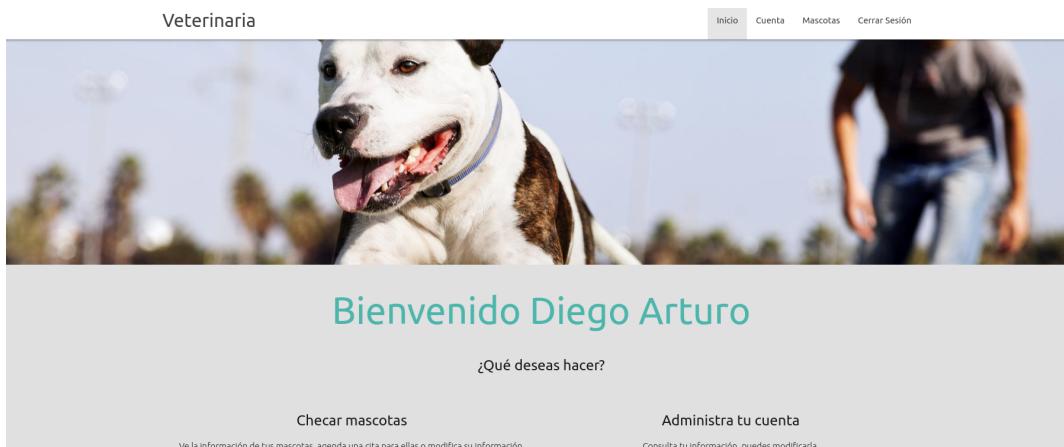


Figura 10.9: Inicio de sesión exitoso a la cuenta con el certificado asignado.

Como podemos darnos cuenta, al querer volver a iniciar sesión nos da acceso de inmediato a nuestra cuenta para este Servicio Web, debido a que anteriormente se había asignado el mismo certificado a la cuenta que se le dio acceso.

10.3. Tiempos de ejecución.

En esta sección mostraremos el tiempo de ejecución de los algoritmos desarrollados para el Prototipo 2, así como la comparación con el Prototipo 1 para demostrar la mejoría que hubo. Todas las pruebas fueron realizadas en igualdad de condiciones, es decir, la computadora recién prendida (sin ningún programa abierto) y sólo ejecutando el algoritmo.

10.3.1. Algoritmo de Chaffing.

Se realizó una prueba al algoritmo de *Chaffing*, el cual incluye: creación de patrón, proceso de chaffing y cifrado de patrón, todo del prototipo 2. Para

la prueba se utilizó una función de JavaScript llamada *performance.now()*, la cual mide el tiempo con una precisión de milisegundos.

Tiempo de ejecución: 17.43 ms.

Si comparamos el tiempo de ejecución del algoritmo de *Chaffing* del Prototipo 1, el cual era de 434.2925 ms, podemos ver que la mejoría fue de 416.8625ms gracias, principalmente, a la disminución del tamaño del patrón (que afecta directamente a la manera en que se crea y en que se recorre para hacer el Chaffing) y a la eliminación del cifrado por bloques AES.

10.3.2. Algoritmo de Winnowing.

Se realizó una prueba al algoritmo de *Winnowing*, el cual incluye: descifrado del patrón y proceso de winnowing. Para la prueba se utilizó una función de Java llamada *System.currentTimeMillis()*, la cual mide el tiempo con una precisión de milisegundos.

Tiempo de ejecución: 12.46 ms.

Si comparamos el tiempo de ejecución del algoritmo de *Winnowing* del Prototipo 1, el cual era de 53.99ms, podemos ver que la mejoría fue de 41.53ms gracias a la disminución del tamaño del patrón y a la eliminación del descifrado por bloques AES.

10.3.3. Inicio de sesión.

Se realizaron un total de 100 pruebas del inicio de sesión por medio de este Trabajo Terminal. Este proceso incluye el tiempo que transcurre desde que el usuario da click en el botón de iniciar sesión en el Servicio Web, es decir, la interceptación de la petición por parte del Componente I) hasta la impresión de la respuesta del Servicio Web, es decir, el inicio de sesión completado exitosamente mostrando la pantalla de inicio del Servicio Web. Para medir el tiempo, se utilizó la función de JavaScript llamada *performance.now()*, al igual que en la medición del algoritmo de Chaffing, puesto que esta medición se realiza desde el Componente I. Extensión.

El tiempo que se muestra a continuación es el promedio de 100 inicios de sesión.

Tiempo promedio de ejecución: 335.28 ms.

Si comparamos el tiempo de inicio de sesión del Prototipo 1, el cual era de 1101.68ms, podemos ver que la mejoría fue de 766.4ms con lo cual se comprueba que el Prototipo 2 es una mejora sustancial al sistema desarrollado en este trabajo sin la pérdida de seguridad.

Capítulo 11

Conclusiones

A lo largo del desarrollo de este trabajo, uno de los puntos que concluimos es que no pudimos evitar completamente el uso de autentificación por usuario y contraseña, pero logramos proporcionar una alternativa al inicio de sesión de los servicios web que deseen implementar este método de autentificación.

Otro punto es que se recomienda que aquel servicio web que desee implementar este método cuente con un certificado SSL para no dejar vulnerable al sistema. Además, debido a las modificaciones que se deben de realizar en la lógica de inicio de sesión del servicio web, no fue posible para este trabajo implementarlo en uno que se encuentre en producción, por lo que se desarrolló un servicio web de prueba.

Gracias a que la API nos ayuda a conectar softwares, nos proporciona un medio para la comunicación entre servicio web y autoridad autenticadora, y así mismo llevar a cabo el proceso de Winnowing.

Finalmente, gracias a la metodología implementada, pudimos desarrollar un prototipo funcional que cumplía con nuestro objetivo, además de mejorarlo en cuestión de tiempo de ejecución de los algoritmos, creando así el prototipo final. Uno de los puntos más importantes de esta mejora fue la reducción del tamaño del patrón que se manda hacia el servicio web, cambiando incluso el cifrado que se implementa.

Capítulo 12

Trabajo a futuro.

Como trabajo a futuro se proponen las siguientes modificaciones para crear un nuevo prototipo con base al último desarrollado:

Se pretende implementar un módulo en la extensión que le permita al usuario administrar los servicios en los cuales se tenga asociado un certificado, con el motivo de darle un mejor control e información al usuario sobre sus cuentas utilizadas para este método de autentificación.

Además se tiene contemplado adaptar un mecanismo biométrico para la obtención de certificado en la extensión, esta parte sustituirá el inicio de sesión por usuario y contraseña; se propone que este mecanismo biométrico sea un lector de huella dactilar.

Tenemos considerado también la implementación del algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm) para el cifrado del patrón de Chaffing en el componente de la extensión, esto con el motivo de incrementar la rapidez del cifrado.

Finalmente, como última propuesta se tiene el implementar este método de autentificación en un servicio web en producción para poder observar su desempeño.

Bibliografía

- [1] A. Aguilera Hernández (2014, abril 25), Sugerencias de Seguridad para Sitios Web, [En línea]. Disponible: <https://www.seguridad.unam.mx/historico/documento/index.html?id=1143>. [Último acceso: 15 de febrero del 2019].
- [2] J. R. Aguirre, *Seguridad Informática y Criptografía*, 2da edición, Madrid, España: Universidad Politécnica de Madrid, 2006.
- [3] M. Bellare y A. Boldyreva, "The Security of Chaffing and Winnowing", California, San Diego, 2015. Disponible: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.4853&rep=rep1&type=pdf>. [Último acceso: 5 de mayo del 2019].
- [4] A. Beutelspacher, *Cryptology*, edición 1994, Washington, Estados Unidos de América: Mathematical Association of America, 1994.
- [5] BBVA de México (2018), ¿Es seguro guardar las contraseñas en el navegador? [En línea] Disponible: <https://www.bbva.com/es/seguro-guardar-contrasenas-navegador/> [Último acceso: 15 de noviembre de 2019]
- [6] CCM (2017), El protocolo HTTP, [En línea]. Disponible: <https://es.ccm.net/contents/264-el-protocolo-http>. [Último acceso: 6 Mayo 2019].
- [7] S. Díaz Santiago, "Generacion De Sucesiones Criptográficamente Fuer tes", tesis de Maestría, División de ciencias básicas e ingeniería, UAM, D.F., México, 2005.
- [8] C. C. Espinosa Madrigal (2011, junio 16), Robo de Identidad y Consecuencias Sociales, [En línea]. Disponible: <https://www.seguridad.unam.mx/historico/documento/>

index.html?id=16?fbclid=IwAR0u8WAXORvBxZ3H-aMzlBhd-6o7g8ycS88eRu7nY1t1XVtCufhEcQ7hWDs. [Último acceso: 15 de febrero del 2019].

- [9] T. Fisher (2019), What Is SHA-1 and How Is It Used for Data Verification?, [En línea]. Disponible: <https://www.lifewire.com/what-is-sha-1-2626011>. [Último acceso: 5 mayo 2019].
- [10] J. D. Gauchat, *El gran libro de HTML5, CSS3 y Javascript*, 1ra edición, Barcelona, España: MARCOMBO S.A., 2012.
- [11] S. Goldwasser y S. Micali, "Probabilistic encryption", *Journal of Computer and System Science*, vol. 28, no. 4, pp. 270–299, 1984.
- [12] desarrolloweb.com (2019), Manual de JQuery, [En línea]. Disponible: http://dmaspv.com/files/page/07042011180222_manual%20de%20jquery%20en%20pdf%20desarrollowebcom.pdf. [Último acceso: 2 de noviembre del 2019].
- [13] Google code (2019), crypto-js, [En línea]. Disponible: <https://code.google.com/archive/p/crypto-js/>. [Último acceso: 5 de mayo del 2019].
- [14] IBM Community (2019), TRIRIGA Wiki Home, [En línea]. Disponible: <https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20TRIRIGA1>. [Último acceso: 5 mayo 2019].
- [15] IBM (2019), IBM TRIRIGA Application Platform, [En línea]. Disponible: https://www.ibm.com/support/knowledgecenter/es/SSHEB3_3.6.0/com.ibm.tap.doc/sso_topics/t_ctr_authenticate.html. [Último acceso: 5 mayo 2019].
- [16] J. Pacheco (2018, julio 18), Entre cookies y privacidad, Oficina de Seguridad del Internauta, [En línea]. Disponible: <https://www.osi.es/es/actualidad/blog/2018/07/18/entre-cookies-y-privacidad>. [Último acceso: 5 de mayo del 2019].
- [17] A. Komarova y A. Menshchikov, "Comparison of Authentication Methods on Web Resources" en Proceedings of the Second International Scientific Conference "Intelligent Information Technologies for Industry", Varna, Bulgaria, 14–16 Septiembre, 2017, pp 106-120.

- [18] J. Larkin, "Implementation of Chaffing and Winnowing: providing confidentiality without encryption", tesis de Licenciatura en Ciencias de la Computación, University of Bath Bath, Inglaterra, 2006. [En línea]. Disponible: <https://pdfs.semanticscholar.org/5520/1a43a747f4a3fb554f241c7b7bc7636b4a07.pdf>. [Último acceso: 5 de mayo del 2019].
- [19] A. Maiorano, *Criptografia: Tecnicas De Desarrollo Para Profesionales*, 1ra edición, D.F., México: Alfaomega, 2009.
- [20] A. Maurya1, P. Kumar Saini y N. Goel, "Chaffing and Winnowing without using steganography and encryption technique", *International Journal of Information Technology and Knowledge Management*, vol. 4, no. 4, pp 515-517, Diciembre, 2011
- [21] A. J. Menezes, P. v. Oorschot y S. Vanstone, *Handbook of Applied Cryptography*, 1ra edición, Florida, Estados Unidos de América: CRC Press, 1996.
- [22] Mozilla (2019, mayo 2), JavaScript. [En línea]. Disponible: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Último acceso: 5 de mayo del 2019].
- [23] Mozilla, (2019, 19 marzo), FileSystem, [En línea]. Disponible: <https://developer.mozilla.org/es/docs/Web/API/FileSystem>. [Último acceso: 5 de mayo del 2019].
- [24] R. S. Pressman, *Ingeniería del Software. Un enfoque práctico*, 7ma edición, D.F, México: McGraw-Hill, 2010.
- [25] R. L. Rivest, "Chaffing and Winnowing: Confidentiality without Encryption", Laboratorio de ciencias de la computación del MIT, Massachusetts, Estados Unidos de América, 1998.
- [26] A. Romero Mier y Terán y M. A. Vasquéz Martínez (2016, abril 19), Aspectos Básicos de la Seguridad en Aplicaciones Web. [En línea]. Disponble: <https://www.seguridad.unam.mx/historico/documento/index.html-id=17>. [Último acceso: 5 de mayo del 2019].
- [27] Universidad Politécnica de Madrid (2004), Criptografía, [En línea]. Disponible: http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/criptografia.html. [Último acceso: 5 de mayo del 2019].

- [28] MongoDB. (2019), ¿Qué es MongoDB?, [En línea]. Disponible: <https://www.mongodb.com/es>. [Último acceso: 2 de noviembre del 2019].
- [29] VeriSign Inc. (2019), Todo lo que debe saber sobre certificados SSL, [En línea]. Disponible: https://www.verisign.com/es_LA/website-presence/online/ssl-certificates/index.xhtml. [Último acceso: 5 de mayo del 2019].
- [30] Wireshark (2019), What is Wireshark?, [En línea]. Disponible: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs. [Último acceso: 20 abril 2019].
- [31] Kerberos (2019), Descripción del Kerberos un servicio de autenticación para los sistemas de red abierta, [En línea]. Disponible: https://www.cisco.com/c/es_mx/support/docs/security-vpn/kerberos/16087-1.html#whatisit
- [32] Van Tilborg, Henk C.A, "Encyclopedia of Cryptography and Security". *Eindhoven University of Technology The Netherlands*, United States of America, Editorial Springer, 2005, 671 pag.
- [33] L. Hernández Encinas (2015), Las Firmas y los Certificados electrónicos (de la Administración Pública del Estado-CSIC) [En línea]. Disponible: <http://www.itefi.csic.es/sites/default/files/hernandez-encinas/firma-y-certificado-electronico.pdf> [Último acceso: 6 noviembre 2019]
- [34] Universitat de Valéncia (2016), ¿Qué son las cookies? [En línea]. Disponible: <https://www.uv.es/uvweb/universidad/es/politica-privacidad/politica-cookies/son-cookies-1285919089226.html> [Último acceso: 6 noviembre 2019]
- [35] J. C. Teague, *Speaking in Styles: Fundamentals of CSS for Web Designers*. 1ra edición. California EEUU: New Riders, 2009.
- [36] Bootstrap (2019), About Bootstrap [En línea]. Disponible: <https://getbootstrap.com/docs/4.3/about/overview/> [Último acceso: 6 noviembre 2019]
- [37] Node.js Foundation (2019), Acerca de Node.js [En línea]. Disponible: <https://nodejs.org/es/about/> [Último acceso: 6 de noviembre 2019]
- [38] Noteworthy Programming Masterpiece (2019), openssl-nodejs [En línea]. Disponible: <https://www.npmjs.com/package/openssl-nodejs> [Último acceso: 6 noviembre 2019]

- [39] D. Bell, M. Parr, *Java para estudiantes*. 3ra edición. México: Pearson Education, 2003.
- [40] Walls, Craig, *Spring in Action*. 5ta edición. EEUU: Manning, 2018.
- [41] The Apache Software Foundation (2019), Apache Tomcat [En línea]. Disponible: <http://tomcat.apache.org/index.html> [Último acceso: 6 noviembre 2019]
- [42] Oracle (2018), MySQL [En línea]. Disponible: <https://www.oracle.com/mysql/> [Último acceso: 6 noviembre 2019]
- [43] Tecnicatura de gestión universitaria (2008), Modelo relacional. Conceptos básicos y fundamentos [En línea]. Disponible: <http://oftgu.eco.catedras.unc.edu.ar/unidad-3/sistemas-de-gestion-de-base-de-datos/modelo-relacional-conceptos-basicos-y-fundamentos/> [Último acceso: 6 de noviembre 2019]
- [44] Oracle (2010), Package java.security [En línea]. Disponible: https://docs.oracle.com/javase/7/docs/api/java/security/package-summary.html#package_description [Último acceso: 6 noviembre 2019]
- [45] Security Appstop (2012), About vsftpd [En línea]. Disponible: <https://security.appspot.com/vsftpd.html#about> [Último acceso: 10 de noviembre 2019]
- [46] Johannes A. Buchmann, Evangelos Karatsiolis, A.W.: Introduction to Public Key Infrastructures. Springer (2013)
- [47] Hostalia whitepapers (2010), ¿Sabes cómo utilizar el protocolo FTP? [En línea]. Disponible: <https://www.hostalia.com/news/noviembre10/sabes-como-utilizar-el-protocolo-FTP.pdf> [Último acceso: 19 de octubre de 2019]
- [48] E-Reding (2001), Protocolo HTTP [En línea]. Disponible: <http://bibing.us.es/proyectos/abreproj/11372/fichero/Memoria%252F05+-+Protocolo+HTTP.pdf> [Último acceso: 21 de octubre de 2019]
- [49] IBM (2018), Protocolo trivial de transferencia de archivos [En línea]. Disponible: https://www.ibm.com/support/knowledgecenter/es/ssw_ibm_i_73/rzal5/rzal5overview.htm [Último acceso: 28 de octubre de 2019]

- [50] EcuRed (2011), Protocolo SMTP [En línea]. Disponible: https://www.ecured.cu/Protocolo_SSMTP [Último acceso: 05 de noviembre de 2019]
- [51] Ayuda de Search de Console (2019), Proteger sitios web con el protocolo HTTPS [En Línea]. Disponible: <https://support.google.com/webmasters/answer/6073543?hl=es-419> [Último acceso: 06 de noviembre de 2019]