



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación de mensajería P2P cifrada y descentralizada

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jesús Ródenas Huerta

Tutor: José Ismael Ripoll Ripoll

Curso 2019-2020

Resum

Actualment, les aplicacions de missatgeria instantània no permeten l'enviament p2p de missatges entre usuaris. Aquests són enviats a un servidor intermediari on són emmagatzemats a l'espera de poder ser enviats al destinatari. El problema que es pretén abordar és l'eliminació d'aqueix servidor intermediari. D'altra banda, les aplicacions existents que intenten resoldre aqueix problema tenen fallades a nivell de privacitat i seguretat.

És per això en aquest treball es durà a terme una nova aplicació de missatgeria instantània punt a punt, capaç de proporcionar confidencialitat, autenticitat i integritat durant la transmissió dels missatges. A més, assegurarà la privacitat de les dades de l'usuari que es troben emmagatzemats a nivell local.

En conclusió, s'ha desenvolupat tant un servidor d'identificadors, emprat perquè diferents instàncies de l'aplicació puguen comunicar-se entre si, com una aplicació, que proporciona *Perfect Forward Secrecy* per a la transmissió dels missatges i el xifrat per a totes les dades de l'usuari emmagatzemades a la base de dades.

Paraules clau: criptografia, p2p, missatgeria

Resumen

Actualmente, las aplicaciones de mensajería instantánea no permiten el envío p2p de mensajes entre usuarios. Estos son enviados a un servidor intermediario donde son almacenados a la espera de poder ser enviados al destinatario. El problema que se pretende abordar es la eliminación de ese servidor intermediario. De otro lado, las aplicaciones existentes que intentan resolver ese problema tienen fallos a nivel de privacidad y seguridad.

Es por ello en este trabajo se llevará a cabo una nueva aplicación de mensajería instantánea punto a punto, capaz de proporcionar confidencialidad, autenticidad e integridad durante la transmisión de los mensajes. Además, asegurará la privacidad de los datos del usuario que se encuentren almacenados a nivel local.

En conclusión, se ha desarrollado tanto un servidor de identificadores, empleado para que diferentes instancias de la aplicación puedan comunicarse entre sí, como una aplicación, que proporciona *Perfect Forward Secrecy* para la transmisión de mensajes y el cifrado para todos los datos del usuario almacenados en la base de datos.

Palabras clave: criptografía, p2p, mensajería

Abstract

Nowadays, instant messaging applications do not allow sending p2p messages between users. These are sent to an intermediary server where they are stored waiting to be sent to the recipient. The problem to be tackled is the elimination of the intermediary server. On the other hand, existing applications that try to solve this problem have privacy and security flaws.

That is why in this work a new point-to-point instant messaging application will be carried out, capable of providing confidentiality, authenticity and integrity during the transmission of messages. In addition, it will ensure the privacy of user data that is stored locally.

In conclusion, both an identifier server, used so that different instances of the application can communicate with each other, and an application that provides Perfect Forward Secrecy for the transmission of messages and encryption for all user data stored in the database.

Key words: criptography, p2p, texting

Índice general

Índice general	V
Índice de figuras	IX
Índice de tablas	XI
<hr/>	
1 Introducción	1
1.1 Motivación	2
1.2 Objetivos	2
1.3 Impacto esperado	2
1.4 Estructura	2
2 Estado del arte	3
2.1 Discusión	4
2.2 Propuesta	5
3 Análisis y diseño de la solución	7
3.1 Análisis de requisitos	7
3.1.1 Requisitos funcionales	7
3.1.2 Requisitos no funcionales	8
3.2 Casos de uso	8
3.2.1 Registrar un usuario	8
3.2.2 Iniciar sesión : Válida	9
3.2.3 Iniciar sesión : Inválida	9
3.2.4 Registrar un contacto	9
3.2.5 Eliminar un contacto	9
3.2.6 Modificar un contacto	10
3.2.7 Buscar contacto	10
3.2.8 Bloquear y desbloquear un contacto	10
3.2.9 Exportar contactos	11
3.2.10 Importar contactos	11
3.2.11 Bloquear un usuario	11
3.2.12 Desbloquear un usuario	11
3.2.13 Permitir o prohibir mensajes de desconocidos	12
3.2.14 Buscar usuario bloqueado	12
3.2.15 Difundir un mensaje	12
3.2.16 Establecer una conversación	13
3.2.17 Restablecer una conversación	13
3.2.18 Envío de mensajes	13
3.2.19 Buscar un mensaje	13
3.2.20 Buscar una conversación	14
3.2.21 Enviar un archivo	14
3.2.22 Difusión de un mensaje a un contacto	14
3.2.23 Cambio de modo de empleo de la aplicación	14
3.3 Lenguaje de programación	15
3.4 Transmisión de mensajes	15

3.4.1	Protocolos de transporte de red	15
3.4.2	Protocolo de control de transmisión	15
3.4.3	Estructura de mensajes	16
3.4.4	Disponibilidad de los clientes para el envío de mensajes	17
3.4.5	Seguridad	21
3.4.6	Solución final	29
3.5	Permitir la recepción de mensajes de desconocidos	29
3.6	Obtención de la ubicación de los usuarios	30
3.6.1	Solicitud manual	30
3.6.2	Servidor para la resolución de identificadores	31
3.6.3	Solución escogida	31
3.6.4	Seguridad	31
3.6.5	Solución final	32
3.7	Redirección de puertos	32
3.7.1	Redirección manual de puertos	32
3.7.2	UPnP	33
3.7.3	DMZ	34
3.7.4	Solución escogida	34
3.8	Almacenamiento de información	34
3.8.1	Aplicación	34
3.8.2	Servidor	38
3.9	Autenticación de los usuarios	39
3.9.1	Seguridad	40
3.10	Interfaz	40
3.10.1	Aplicación	40
3.10.2	Servidor	42
4	Implementación	43
4.1	Diffie-Hellman y AES-256 (CBC)	43
4.2	Envío y recepción de archivos	45
4.3	Generación del identificador de los usuarios	46
4.4	Generación y comprobación del <i>hash</i> del usuario	46
4.5	Autenticación entre clientes	47
4.6	AES	48
5	Implantación	51
5.1	Instalación de Java	51
5.1.1	Windows	51
5.1.2	Ubuntu	51
5.2	Librerías	52
5.2.1	SQLite	53
5.2.2	JSON	53
5.2.3	WeUPnP	53
6	Validación	55
6.1	Direccionamiento de puertos mediante UPnP	55
6.2	Listas blanca y negra del servidor de identificadores	55
6.3	Análisis del tráfico	57
6.4	Cifrado de la base de datos	58
7	Conclusiones	63
7.1	Cumplimiento de objetivos	63
7.2	Relación del trabajo desarrollado con los estudios cursados	63
7.3	Trabajos futuros	64
	Bibliografía	65

Apéndice

A Código de la implementación	69
A.1 Conversor de un conjunto de <i>bytes</i> a hexadecimal	69
A.2 Método envío de un conjunto de <i>bytes</i>	69
A.3 Método recepción de un conjunto de <i>bytes</i>	70
A.4 Método envío mensaje cifrado	70
A.5 Método de recepción de mensaje cifrado	71
A.6 Generadores de caracteres aleatorios	71
A.6.1 RandomString	71
A.6.2 RandomAlfaNumericaString	72
A.6.3 RandomInt	72
A.7 Generación de fecha	72
A.8 Cifrado de la base de datos	73
A.8.1 Insertar un contacto	73
A.8.2 Descifra contacto	74
A.8.3 Inserta mensaje	75
A.8.4 Obtén mensaje	75
A.8.5 Inserción de un mensaje temporal	76
A.8.6 Obtén mensajes temporales	76
A.8.7 Inserta identificador bloqueado	77
A.8.8 Obtén identificadoras bloqueados	77

Índice de figuras

1.1	Alfabeto para el cifrado de mensajes en 4º de primaria.	1
2.1	Jami. Capturas de pantallas.	3
2.2	Ricochet. Ejemplo de chat.	4
2.3	Dust menú.	4
3.1	Servidor SMTP UPV.	16
3.2	Ejemplo código XML.	17
3.3	Ejemplo código JSON.	17
3.4	Colas de mensajería sin gestor.	18
3.5	Fase 1. Transmisión de mensajes.	19
3.6	Fase 2. Reenvío de mensajes.	20
3.7	Password Authentication Protocol.	21
3.8	Challenge-handshake authentication protocol.	22
3.9	Diffie-Hellman Key Exchange.	23
3.10	Aproximación a la generación de claves públicas con curvas elípticas.	24
3.11	Gráfica de tiempos de ECDH y DH.	24
3.12	Algoritmo de cifrado AES modo ECB.	25
3.13	Algoritmo de cifrado AES modo CBC.	26
3.14	Algoritmo de cifrado AES modo CFB.	26
3.15	Algoritmo de cifrado AES modo OFB.	27
3.16	Algoritmo de cifrado AES modo CTR.	27
3.17	Gráfica de la comprobación de integridad de los mensajes.	28
3.18	Pedir una dirección IP y un puerto mediante otra aplicación.	30
3.19	Servidor de identificadores. Solicitud de IP.	31
3.20	Reenvío de puertos manual.	33
3.21	UPnP rúter dd-wrt.	33
3.22	DMZ rúter ZTE.	34
3.23	Sistema de almacenamiento. Árbol de ficheros	35
3.24	Sistema de almacenamiento. Base de datos	36
3.25	Sistema de almacenamiento. Árbol de ficheros, solución final.	36
3.26	Sistema de almacenamiento. Base de datos, solución final.	37
3.27	Sistema de almacenamiento. Árbol de ficheros con permisos.	37
3.28	Permisos de los ficheros del servidor de identificadores.	39
3.29	Pantalla de autenticación de usuarios.	39
3.30	Interfaz de Telegrama.	40
3.31	Interfaz de Whatsapp.	41
3.32	Pantallas aplicación.	41
3.33	Pantalla chat aplicación.	42
3.34	Interfaz del servidor de identificadores.	42
4.1	Generación de la clave pública ECDH por parte de Alice.	43
4.2	Generación de la clave pública ECDH por parte de Bob.	44
4.3	Generación del secreto compartido.	44

4.4	Generación de cifrador y descifrador.	44
4.5	Lineas de código para la transmisión de archivos.	45
4.6	Líneas de código para la recepción de archivos.	45
4.7	Generación del identificador de un usuario.	46
4.8	Generación del <i>hash</i> del usuario.	46
4.9	Comprobación del <i>hash</i> del usuario.	47
4.10	Generación del reto.	47
4.11	Respuesta al reto.	48
4.12	Generación de cifrado y descifrador AES.	49
4.13	Método para el cifrado de texto.	49
4.14	Método para el descifrado de texto.	50
5.1	Árbol de fichero de la aplicación con librerías.	53
5.2	Árbol de fichero del servidor con librerías.	53
6.1	Tabla de direccionamiento de puertos con UPnP.	55
6.2	Mensaje de bloqueo de la aplicación.	56
6.3	Servidor de identificadores. Resultado al obtener e informar al servidor.	56
6.4	Registro del servidor de mensajes de una IP no bloqueada.	56
6.5	Lista Blanca. Registro del servidor de mensajes de una IP bloqueada.	56
6.6	Lista Blanca. Registro del servidor de mensajes de una IP no bloqueada.	56
6.7	Análisis de los paquetes transmitidos al servidor de identificadores	57
6.8	Registro del servidor de mensajes.	57
6.9	Captura de pantalla del mensaje enviado al contacto Z.	58
6.10	Análisis de los paquetes transmitidos al cliente Z.	58
6.11	Mensajes enviados al contacto <i>b</i>	59
6.12	Contactos almacenados en la aplicación.	59
6.13	Usuarios bloqueados y almacenados en la aplicación.	59
6.14	Mensajes cifrados y almacenados en la base de datos.	59
6.15	Contactos cifrados y almacenados en la base de datos.	60
6.16	Usuarios cifrados bloqueados y almacenados en la base de datos.	60
6.17	Mensajes temporales cifrados y almacenados en la base de datos.	60
6.18	Conversión del mensaje de base64 a hexadecimal.	60
6.19	Código para mostrar la clave y el IV en formato hexadecimal.	61
6.20	Clave e IV en sistema hexadecimal.	61
6.21	Texto descifrado a través de la web CyberChef.	61
A.1	Implementación del conversor.	69
A.2	Método para el envío de un conjunto de <i>bytes</i>	69
A.3	Método para la recepción de un conjunto de <i>bytes</i>	70
A.4	Método para el envío de un mensaje cifrado.	70
A.5	Método para la recepción de un mensaje cifrado.	71
A.6	Método de generación de una cadena de caracteres de <i>bytes</i> aleatorios.	71
A.7	Método de generación de una cadena de caracteres alfanuméricos aleatorios.	72
A.8	Método de generación de un entero aleatorio.	72
A.9	Método de generación de fecha.	72
A.10	Método para cifrar e insertar un contacto en la base de datos.	73
A.11	Método para la obtención y descifrado de un contacto de la base de datos.	74
A.12	Método para el cifrado y la inserción de un mensaje en la base de datos.	75
A.13	Método para la obtención y descifrado de un mensaje de la base de datos.	75
A.14	Método la inserción de un mensaje temporal (ya cifrado) en la base de datos.	76
A.15	Método para la obtención mensajes temporales de la base de datos.	76

A.16 Método para el cifrado e inserción de un identificador, que ha sido bloqueado, en la base de datos.	77
A.17 Método para la obtención y descifrado de identificadores, que han sido bloqueado, de la base de datos.	77

Índice de tablas

3.1 Tabla de tiempos de ECDH y DH.	25
3.2 Tabla de la comprobación de integridad de los mensajes.	29

CAPÍTULO 1

Introducción

Todos hemos sido niños alguna vez y hemos sentido la necesidad de contarle algo a un compañero en pleno transcurso de una clase. En ese momento, a diferencia de los niños de ahora, nuestra única opción era escribir un mensaje en un trozo de papel. Ese trozo de papel con nuestros más secretos pensamientos debía ser doblado cuidadosamente y ser pasado de un compañero a otro a través de una red de alumnos estratégicamente seleccionada para que llegase al fin a su destinatario final. Y todos recordamos la tensión que pasábamos para que ese mensaje no fuese descubierto por la persona que daba la lección en la pizarra o abierto y leído por uno de los compañeros que lo transportaban. Los más creativos y desconfiados de nosotros fuimos capaces de desarrollar un código secreto, nuestro propio alfabeto rúnico al más puro estilo vikingo, con el que nuestros mensajes serían indescifrables para los curiosos.



Figura 1.1: Alfabeto para el cifrado de mensajes en 4° de primaria.

La realidad es que en la actualidad las cosas no distan tanto de esos pequeños que escribían notas durante las clases. Ahora enviamos los mismos mensajes a nuestros amigos y parejas, solo que en la actualidad estas notas pasan a ser una retahíla de unos y ceros que corren por la red, pasando por servidores y nodos en lugar de entre nuestros compañeros de aula, para llegar a esa persona a la que queremos contactar. Y al igual que en primaria los más originales inventábamos nuestros alfabetos, algunas de las aplicaciones que utilizamos cifran nuestros mensajes para hacer más difícil la tarea a los curiosos.

Es también una realidad que siempre fue lo más seguro dar el mensaje en mano a nuestro compañero, y de la misma forma sucede con las comunicaciones digitales. Resulta más seguro enviar las misivas de un par a otro, que pasar por un intermediario. Estos intermediarios ahora son servidores que sufren ataques a todas horas. Este fue el caso, por ejemplo, del ataque de denegación de servicio distribuido a los servidores de Telegram[1] o los varios casos de fuga de información de Facebook[2][3].

Además, siendo conscientes de que en numerosas ocasiones los servidores dejan de funcionar correctamente debido a una mala configuración de los mismos, como sucedía con Whatsapp el pasado año[4]: ¿No sería mejor evitar que sobre ellos recayese todo el peso del envío de mensajes? y ¿No sería mejor que cada uno pudiese tener el suyo propio?

Como respuesta a estas preguntas, se plantea en este trabajo la viabilidad y realización de una aplicación de mensajería que envíe los mensajes directamente al usuario destinatario, dejando al servidor como utensilio para indicar a la aplicación la dirección del mismo.

1.1 Motivación

La idea del trabajo de fin de grado, surgió a partir de otro trabajo realizado en la asignatura «Criptografía», donde se decidió investigar sobre los métodos de cifra empleados en las aplicaciones de mensajería instantánea Whatsapp y Telegram. Durante el proceso de investigación quedó latente la ausencia de información pública sobre algunos aspectos críticos de la implementación.

Fue por ello por lo que se decidió diseñar una aplicación de mensajería de código abierto, cuya seguridad residiese en las claves empleadas durante el cifrado, y no en la seguridad por oscuridad. Finalmente, se pretende conseguir que todos los mensajes sean realmente entre clientes sin la necesidad de pasar por un servidor central que los almacene y envíe posteriormente al usuario destinatario.

1.2 Objetivos

Desarrollar una aplicación de mensajería p2p gratuita, que dependa lo mínimo posible del uso de servidores centralizados y que pueda ser empleada por usuarios que tienen una cierta destreza en el empleo de aplicaciones de mensajería instantánea. Además, poner en práctica la implementación de diferentes conceptos relacionados con la criptografía y la seguridad.

1.3 Impacto esperado

El usuario final tendrá la confianza de que existe una aplicación de mensajería instantánea cuyos mensajes serán directamente enviados a su destinatario y en ningún momento estarán siendo analizados para el beneficio de terceros.

1.4 Estructura

La memoria de este trabajo se encuentra dividida en cinco partes. En la primera parte se examinan las diferentes aplicaciones p2p de mensajería que existen hoy en día en el mercado. En la segunda parte se analizan y se buscan soluciones a los problemas existentes durante el desarrollo de la aplicación. En la tercera parte se comentan los puntos claves de la implementación. En la cuarta parte se describe la instalación del sistema desarrollado para su posterior evaluación. En la quinta parte se comprueba el correcto funcionamiento del sistema. Finalmente, en la sexta parte se sintetizan los puntos más relevantes llevados a cabo en este proyecto.

CAPÍTULO 2

Estado del arte

A día de hoy, las aplicaciones de mensajería instantánea más utilizadas son Whatsapp Messenger, Line y Telegram, con **110 millones**, **12 millones** y **5 millones** de descargas respectivamente en la Play Store. Estas aplicaciones presentan una característica común: su diseño es centralizado, es decir, todo el tráfico de las mismas pasa por sus servidores antes de llegar a destino. El uso de esta estructura implica que una vulnerabilidad en uno de sus servidores centrales suponga la fuga de grandes cantidades de información de los usuarios registrados en ellas.

Por esta razón, a lo largo del tiempo han ido surgiendo aplicaciones de mensajería basadas en redes de pares para evitar que surjan este tipo de problemas.

Entre todas ellas cabe destacar las siguientes aplicaciones.

1. **Jami**¹ es una aplicación multiplataforma que destaca por ser un proyecto GNU, respaldado por la *Free Software Foundation*, distribuida bajo la licencia GPLv3 y que cumple con el estándar X.509, un estándar para infraestructuras de claves públicas. Además, hoy en día sigue manteniendo soporte para todas las plataformas.

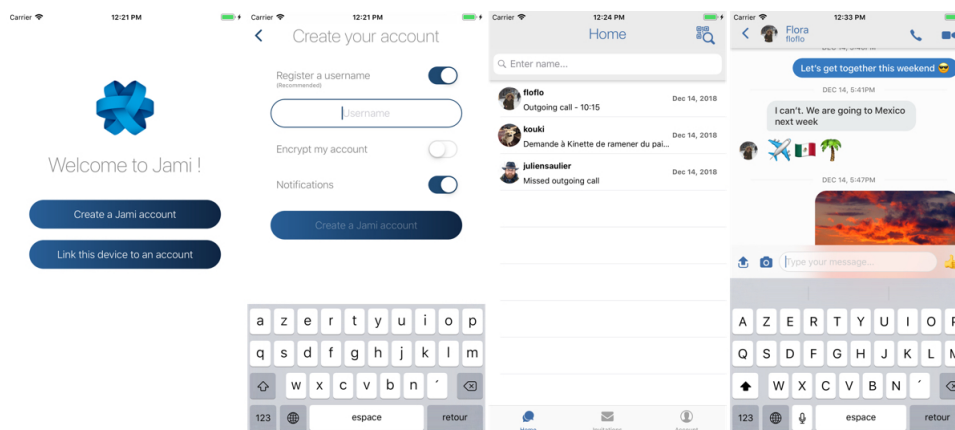


Figura 2.1: Jami. Capturas de pantallas.

2. **Ricochet**² es una aplicación disponible para Windows, Mac y Linux, cuya principal característica y por lo que destaca, es el anonimato que proporciona al usuario. Este anonimato se consigue mediante la eliminación de los metadatos a la hora de transmitir archivos y el uso de la red Tor para el establecimiento de conexiones entre usuarios.

¹<https://jami.net/>

²<https://ricochet.im/>

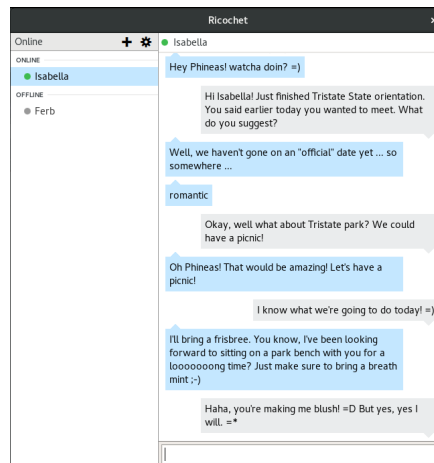


Figura 2.2: Ricochet. Ejemplo de chat.

3. **Dust**³ esta aplicación destaca por eliminar los mensajes cinco segundos después de haber sido leídos, o a las veinticuatro horas en caso de no haber sido abiertos. Además, también es famosa hacer uso de la tecnología de cadenas de bloques para el uso de la transmisión de mensajes [5].

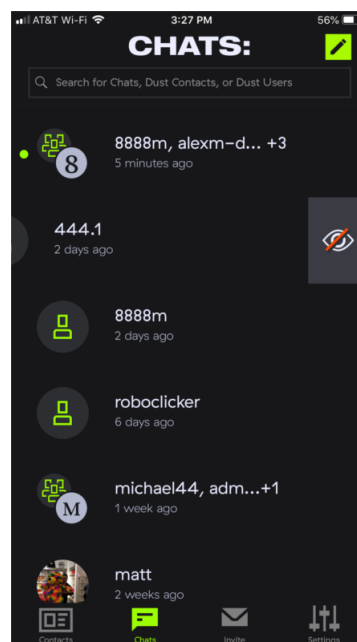


Figura 2.3: Dust menú.

2.1 Discusión

En la sección anterior se hablaba sobre las diferentes aplicaciones p2p de mensajería que existen hoy en día, pero no se ha hablado de los fallos que tienen. En esta sección se discutirá sobre los problemas que tienen dichas aplicaciones:

³<https://usedust.com/>

1. En el caso de **Jami** no se proporciona *Perfect Forward Secrecy*⁴ (**PFS**) para la transmisión de mensajes, « Los mensajes son cifrados con una clave RSA» según se indica en la sección FAQ de su página [web](#). Por consiguiente si un atacante consigue «romper» la criptografía de la clave pública de alguno de los usuarios de la aplicación, este será capaz de leer tanto los mensajes cifrados capturados previamente como los generados por el usuario en un futuro [14].
2. El proyecto **ricochet** dejó de ser actualizado hace tres años, dejando una advertencia en su página web que informa de que Ricochet es un experimento, y pide a los usuarios que por favor «don't risk your safety any more than necessary».
3. El empleo de **Dust** según la web [derechodelared.com](#) «es una buena aplicación de mensajería si quieres evitar que tu destinatario pueda usar los mensajes que le envías en tu contra. Pero nada más.» porque toda tu información como usuario es recopilada por sus servidores a excepción de los mensajes [6].

Finalmente, se quiere incidir en que ninguna aplicación de mensajería anteriormente mencionada es perfecta, ya sea por el desempleo de métodos de cifrado más robustos a la hora de cifrar los mensajes, la falta de soporte o finalmente, la recopilación de información generada por el usuario para su posterior distribución a terceros [7].

2.2 Propuesta

Con el fin de evitar el uso de servidores de mensajería empleados en las aplicaciones actuales de mensajería, y además evitar que diferentes mensajes sean cifrados con una misma clave, como es el caso de algunas de las aplicaciones comentadas anteriormente. Se propone como solución, la creación de una aplicación de mensajería p2p que evite en mayor medida la centralización del envío de mensajes almacenando todos los datos generados por el usuario en el lado del cliente. Finalmente, esta aplicación proporcionará *Perfect Forward Secrecy* para el envío de mensajes entre usuarios.

⁴Es una propiedad de algunos sistemas criptográficos que garantiza que el descubrimiento de una clave de cifrado no compromete la seguridad de las claves empleadas anteriormente (Se detallará en secciones posteriores).

CAPÍTULO 3

Análisis y diseño de la solución

Este capítulo sintetiza: los requisitos necesarios para el desarrollo de la aplicación, el análisis de los problemas existentes durante el desarrollo de la aplicación y las soluciones a tratar, teniendo siempre en cuenta el punto de vista de la ciberseguridad.

3.1 Análisis de requisitos

3.1.1. Requisitos funcionales

- La aplicación deberá permitir registrar nuevos usuarios.
- La aplicación deberá mostrar los usuarios registrados para poder iniciar sesión.
- Un usuario deberá iniciar sesión a partir de su usuario y contraseña.
- La aplicación deberá permitir registrar nuevos contactos en la agenda de contactos.
- La aplicación deberá permitir editar contactos.
- La aplicación deberá permitir buscar un contacto en la agenda de contactos.
- La aplicación deberá permitir eliminar contactos.
- La aplicación deberá permitir bloquear y desbloquear contactos.
- La aplicación deberá permitir bloquear y desbloquear usuarios no agregados como contactos.
- La aplicación deberá permitir al usuario enviar mensajes a sus contactos.
- La aplicación deberá permitir al usuario difundir un mensaje entre sus contactos.
- La aplicación deberá permitir al usuario enviar cualquier tipo de archivo a sus contactos.
- La aplicación deberá de informar tanto de forma visual como auditiva la recepción de un nuevo mensaje.
- La aplicación deberá permitir al usuario cambiar el estilo de la aplicación.
- El usuario deberá poder buscar mensajes enviados o recibidos de un contacto específico.

- La aplicación deberá mostrar al usuario si ya ha hablado anteriormente con un contacto.
- La aplicación no deberá permitir que dos usuarios iniciasen sesión sobre la misma cuenta a la vez.
- La aplicación no deberá cifrar más de un mensaje con la misma clave.
- La aplicación deberá reproducir un sonido cada vez que el usuario reciba un mensaje.

3.1.2. Requisitos no funcionales

- Los usuarios registrados deberán ser almacenados localmente.
- Los contactos registrados por el usuario deberán de ser almacenados localmente y de una forma segura.
- Los mensajes y archivos enviados por el usuario deberán de ser enviados a través de medios seguros.
- La aplicación deberá asegurar que los datos estén protegidos frente al acceso no autorizado.
- La aplicación deberá ejecutarse en Windows y Linux.
- La aplicación deberá recuperar los mensajes de una conversación eficientemente.
- La aplicación deberá contar con una interfaz intuitiva.

3.2 Casos de uso

3.2.1. Registrar un usuario

- **Prerrequisito:** El usuario no se ha registrado
- **Pasos:**
 1. El usuario inicia la aplicación.
 2. El usuario selecciona la pestaña de registro.
 3. El usuario escribe el usuario y contraseña que desea tener.
 4. El usuario escribe la dirección IP o dominio del servidor de identificadores.
 5. El usuario escribe el puerto del servidor de identificadores.
 6. El usuario pulsa el botón de registrarse.
- **Salida:** La aplicación muestra una ventana indicando al usuario que se ha registrado con éxito.

3.2.2. Iniciar sesión : Válida

- **Prerrequisito: El usuario se ha registrado previamente**

- **Pasos:**

1. La aplicación muestra al usuario un listado con los usuarios registrados en el sistema.
2. El usuario selecciona uno de los usuarios.
3. El usuario escribe la contraseña correctamente.
4. La aplicación comprueba la contraseña.
5. La aplicación muestra al usuario la pantalla para entablar conversaciones.

3.2.3. Iniciar sesión : Inválida

- **Prerrequisito: El usuario se ha registrado previamente**

- **Pasos:**

1. La aplicación muestra al usuario un listado con los usuarios registrados en el sistema.
2. El usuario selecciona uno de los usuarios.
3. El usuario escribe una contraseña diferente a la del registro.
4. La aplicación comprueba la contraseña.
5. La aplicación informa al usuario que las contraseñas no coinciden.

3.2.4. Registrar un contacto

- **Prerrequisito: El usuario ha iniciado sesión.**

- **Pasos:**

1. El usuario pulsa el botón de abrir agenda.
2. La aplicación mostrará una tabla con todos los contactos registrados.
3. El usuario pulsa el botón de «Añadir» contacto.
4. La aplicación muestra una pantalla para que el usuario escriba los campos necesarios para registrar un usuario.
5. El usuario introduce el apodo y el identificar del contacto.
6. El usuario pulsa el botón «Aceptar».
7. La aplicación registra el contacto.

3.2.5. Eliminar un contacto

- **Prerrequisito: Haber registrado un contacto.**

- **Pasos:**

1. El usuario pulsa el botón de abrir agenda.
2. El usuario selecciona al menos un contacto.
3. El usuario pulsa el botón «Borrar».

4. La aplicación muestra un mensaje al usuario para que confirme dicha acción.
5. El usuario pulsa el botón «Aceptar».
6. La aplicación borra toda la información relacionada con el contacto.

3.2.6. Modificar un contacto

- **Prerrequisito: Haber registrado un contacto.**

- **Pasos:**

1. El usuario pulsa el botón de abrir agenda.
2. La aplicación muestra una tabla con todos los contactos registrados.
3. El usuario selecciona un contacto.
4. El usuario pulsa el botón «Modificar».
5. La aplicación muestra al usuario una ventana con los campos con los que se registro al contacto.
6. El usuario modifica los campos que desea cambiar del contacto.
7. El usuario pulsa el botón «Aceptar».
8. La aplicación registra los cambios realizados a los valores del contacto.

3.2.7. Buscar contacto

- **Prerrequisito: Haber iniciado sesión.**

- **Pasos:**

1. El usuario pulsa el botón de abrir agenda.
2. La aplicación muestra una tabla con todos los contactos registrados.
3. El usuario introduce en el campo de búsqueda, el apodo del contacto a buscar.
4. La aplicación muestra en la tabla de contactos, los contactos cuyo apodo coincide con lo escrito por el usuario.

3.2.8. Bloquear y desbloquear un contacto

- **Prerrequisito: Haber registrado un contacto.**

- **Pasos:**

1. El usuario pulsa el botón de abrir agenda.
2. La aplicación muestra una tabla con todos los contactos registrados.
3. El usuario selecciona un contacto.
4. El usuario pulsa el botón «Bloquear» o «Desbloquear».
5. La aplicación muestra o oculta en la fila del contacto, la imagen de un candado.
6. La aplicación registra o elimina el identificador del contacto en la lista de bloqueados.

3.2.9. Exportar contactos

- **Prerrequisito: Haber iniciado sesión.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. La aplicación muestra una tabla con todos los contactos registrados.
 3. El usuario pulsa el botón «Exportar».
 4. La aplicación genera un documento con todos los contactos registrados en el sistema.

3.2.10. Importar contactos

- **Prerrequisito: Haber iniciado sesión.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. La aplicación muestra una tabla con todos los contactos registrados.
 3. El usuario pulsa el botón «Importar».
 4. La aplicación muestra al usuario una pantalla que el usuario empleará para seleccionar el archivo con los contactos a importar.
 5. El usuario seleccionará el archivo con los contactos a importar.
 6. El usuario pulsa el botón «Aceptar».
 7. La aplicación registra todos los contactos que no existían previamente.

3.2.11. Bloquear un usuario

- **Prerrequisito: Haber iniciado sesión.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. El usuario selecciona la pestaña de «Bloqueados».
 3. La aplicación muestra al usuario la lista con todos los usuarios bloqueados.
 4. El usuario pulsa el botón «Añadir».
 5. La aplicación muestra una ventana al usuario para que escriba el identificador del usuario a bloquear.
 6. El usuario escribe el identificador del usuario a bloquear.
 7. El usuario pulsa el botón «Aceptar».
 8. La aplicación registra el identificador del usuario en la lista de bloqueados.

3.2.12. Desbloquear un usuario

- **Prerrequisito: Haber bloqueado un usuario o contacto previamente.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.

2. El usuario selecciona la pestaña de «Bloqueados».
3. La aplicación muestra al usuario la lista con todos los usuarios bloqueados.
4. El usuario selecciona el identificador del usuario a desbloquear.
5. El usuario pulsa el botón «Desbloquear».
6. La aplicación elimina el identificador del usuario de la lista de bloqueados.

3.2.13. Permitir o prohibir mensajes de desconocidos

- **Prerrequisito: Haber iniciado sesión previamente.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir ajustes.
 2. La aplicación muestra una pantalla con los campos de los ajustes.
 3. El usuario pulsa uno de los dos botones de radio para permitir o prohibir la recepción de mensajes de desconocidos.
 4. El usuario pulsa el botón «Aceptar».
 5. La aplicación guarda los cambios y muestra un mensaje indicando al usuario que reinicie la aplicación para que los cambios tengan efecto.

3.2.14. Buscar usuario bloqueado

- **Prerrequisito: Haber iniciado sesión.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. El usuario selecciona la pestaña de «Bloqueados».
 3. La aplicación muestra al usuario la lista con todos los usuarios bloqueados.
 4. El usuario escribe los caracteres a buscar en la lista de bloqueados.
 5. La aplicación muestra en la lista de bloqueados los identificadores de contacto que coinciden con los caracteres escritos por el usuario.

3.2.15. Difundir un mensaje

- **Prerrequisito: Haber registrado un contacto.**
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. La aplicación mostrará una tabla con todos los contactos registrados.
 3. El usuario selecciona todos los contactos con los que se quiere compartir un mismo mensaje.
 4. El usuario pulsa el botón «Difundir».
 5. La aplicación muestra al usuario una pantalla donde el usuario puede escribir el mensaje a enviar.
 6. El usuario escribe el mensaje a difundir.
 7. El usuario pulsa el botón «Aceptar».
 8. La aplicación procede a difundir el mensaje.

3.2.16. Establecer una conversación

- **Prerrequisito:** Haber registrado un contacto.
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. La aplicación mostrará una tabla con todos los contactos registrados.
 3. El usuario selecciona un contacto.
 4. El usuario pulsa el botón «Hablar».
 5. La aplicación muestra al usuario una pantalla con todos los mensajes enviados y recibidos, y un campo donde puede escribir los mensajes que se desean enviar.
 6. La aplicación crea una fila en la tabla de conversación.

3.2.17. Restablecer una conversación

- **Prerrequisito:** Haber establecido una conversación.
- **Pasos:**
 1. El usuario pulsa en una de las conversaciones realizadas previamente, mostradas en la tabla de conversaciones.
 2. La aplicación restaura la conversación, mostrando los mensajes enviados y recibidos.

3.2.18. Envío de mensajes

- **Prerrequisito:** Haber establecido una conversación.
- **Pasos:**
 1. El usuario escribe el mensaje a enviar en el campo de texto.
 2. El usuario pulsa el botón de enviar.
 3. La aplicación obtiene el mensaje escrito por el usuario y lo envía al contacto.

3.2.19. Buscar un mensaje

- **Prerrequisito:** Haber establecido o haber restablecido una conversación.
- **Pasos:**
 1. El usuario escribe en la barra de búsqueda un fragmento de texto a buscar.
 2. La aplicación muestra al usuario todos los mensajes que contienen dicho fragmento.

3.2.20. Buscar una conversación

- **Prerrequisito:** Haber iniciado sesión.
- **Pasos:**
 1. El usuario introduce el apodo del contacto en el campo de búsqueda de conversaciones.
 2. La aplicación muestra al usuario las conversaciones donde se encuentra el contacto.

3.2.21. Enviar un archivo

- **Prerrequisito:** Haber establecido o restablecido una conversación.
- **Pasos:**
 1. El usuario pulsa el botón de enviar un fichero.
 2. La aplicación muestra una pantalla de navegación para seleccionar el archivo a enviar.
 3. El usuario selecciona el archivo a enviar.
 4. El usuario pulsa el botón de aceptar.
 5. La aplicación envía el archivo al contacto.

3.2.22. Difusión de un mensaje a un contacto

- **Prerrequisito:** Haber registrado al menos contacto.
- **Pasos:**
 1. El usuario pulsa el botón de abrir agenda.
 2. El usuario selecciona al menos un contacto.
 3. El usuario pulsa el botón «Difundir».
 4. La aplicación muestra una pantalla al usuario donde puede escribir el texto del mensaje a enviar.
 5. El usuario escribe el mensaje a enviar.
 6. El usuario pulsa el botón «Aceptar».
 7. La aplicación envía el mensajes a los contactos seleccionados.

3.2.23. Cambio de modo de empleo de la aplicación

- **Prerrequisito:** Haber iniciado sesión.
- **Pasos:**
 1. El usuario pulsa el botón de ajustes.
 2. La aplicación muestra una pantalla con los campos de los ajustes.
 3. El usuario elige si desea emplear la aplicación a nivel local o la quiere emplear a través de Internet.
 4. El usuario pulsa el botón «Aceptar».
 5. La aplicación guarda los cambios y muestra un mensaje indicando al usuario que reinicie la aplicación para que los cambios tengan efecto.

3.3 Lenguaje de programación

Antes de empezar a implementar cualquier aspecto de la aplicación es necesario elegir el lenguaje que se va emplear para su implementación. Hoy en día existe una gran variedad de lenguajes de programación como Python, Java, JavaScript, C, C++, C#, Golang, etc. Todos ellos son igualmente válidos para desarrollar cualquier tipo de aplicación. No obstante, se ha elegido Java 8 como lenguaje de programación, porque el autor tiene una cierta experiencia desarrollando programas, evitando aprender un lenguaje nuevo, acortando el tiempo de implementación y desarrollo de la aplicación. Además, las librerías por defecto de Java cuentan con los métodos necesarios para desarrollar interfaces, crear conexiones a través de la red, cifrar y almacenar datos generados por la aplicación.

Finalmente, cabe decir que se empleará la versión 8 de Java porque evita que los usuarios tengan que instalarse JavaFX¹ [8], debido a que a partir de Java 11, JavaFX funciona como un módulo independiente de Java JDK, dificultando a los usuarios finales la puesta en marcha de la aplicación.

3.4 Transmisión de mensajes

En esta sección se analizan los problemas relacionados con la transmisión de mensajes, aportando y evaluando diferentes soluciones tanto a nivel funcional como a nivel de la seguridad.

3.4.1. Protocolos de transporte de red

Para empezar, al desarrollar un software que necesita comunicación entre nodos, se tiene que determinar el protocolo de transporte que mejor se adapta a dicho software.

3.4.2. Protocolo de control de transmisión

El protocolo de control de transmisión o en inglés *Transfer Control Protocol (TCP)* [9] es un protocolo diseñado para transmitir información entre nodos y para verificar su correcta recepción, puesto que el protocolo garantiza que todos los datos se transmitan correctamente, ya que es capaz de detectar errores de transmisión a partir del *checksum*², y que el receptor pueda ensamblarlos en el orden correcto, debido al uso de números de secuencia. Además, el protocolo TCP generalmente se usa junto con el Protocolo de Internet (IP) y se conoce comúnmente como la pila de protocolos TCP / IP.

Protocolo de datagramas de usuario

El protocolo de datagramas de usuario o en inglés *User Datagram Protocol (UDP)* [10] tiene un funcionamiento similar al protocolo TCP, pero, al igual que el protocolo IP, no es un protocolo de transporte orientado a conexión. Es decir, el protocolo UDP no comprueba que los segmentos enviados hayan sido recibidos. Haciendo que la comprobación de que todos los paquetes hayan sido recibidos resida en capas superiores.

¹Permite a los desarrolladores integrar gráficos vectoriales, animación, sonido y activos web de vídeo en una aplicación interactiva, completa y atractiva

²Campo ubicado en la cabecera del segmento TCP, resultado de calcular la suma en complemento a uno del contenido de la cabecera y datos del segmento TCP

La principal ventaja del protocolo UDP consiste en la velocidad de transmisión, pues se prescinde de la comprobación de los datos recibidos aumentando su velocidad de transmisión.

Solución escogida

Para el desarrollo de la aplicación se ha escogido el protocolo de control de transmisión (TCP) porque comprueba la correcta recepción de los mensajes, algo necesario en una aplicación de mensajería.

3.4.3. Estructura de mensajes

Durante el proceso de diseño de una aplicación de mensajería es necesario plantearse que estructura ha de tener el mensaje a enviar, pues tiene que ser entendido por el receptor una vez lo haya recibido. Se pone como ejemplo, el caso de **Simple Mail Transfer Protocol** (SMTP) donde cada campo atributo-valor es separado por «:», a excepción del texto del mensaje que carece de campo atributo-valor.

```
220 smtp.upv.es ESMTP Sendmail 8.14.7/8.14.7; Mon, 25 May 2020 19:35:29 +0200
> HELO smtp.upv.es
250 smtp.upv.es Hello 38.red-79-147-254.dynamicip.rima-tde.net [79.147.254.38],
pleased to meet you
> MAIL FROM: jerohue@etsinf.upv.es
250 2.1.0 jerohue@etsinf.upv.es... Sender ok
> RCPT TO: jerohue@etsinf.upv.es
250 2.1.5 jerohue@etsinf.upv.es... Recipient ok
> DATA
354 Enter mail, end with "." on a line by itself
> SUBJECT: Esto es una prueba
Este es el contenido del mensaje.
Puede contener varias líneas.
>.
250 2.0.0 04PHZTpe010065 Message accepted for delivery
```

Figura 3.1: Servidor SMTP UPV.

Con el fin de no tener que desarrollar un nuevo lenguaje de marcado, pues supondría la realización de análisis, pruebas de diseño, evaluaciones de rendimiento y búsquedas de debilidades y/o vulnerabilidades en la implementación de los serializadores de datos; alejándose del desarrollo de este trabajo. Se han propuesto diferentes estructuras de datos empleadas actualmente en el mundo laboral y de la investigación, que además ofrecen sus propias librerías para evitar su implementación al programador.

Lenguaje de marcado extensible

El lenguaje de marcado extensible o en inglés *eXtensible Markup Language* (XML) [12] es un metalenguaje que fue diseñado para estructurar, almacenar y para intercambiar datos entre diferentes aplicaciones. Se convirtió en un estándar, ya que es extensible³ y puede ser utilizado por cualquier aplicación independientemente de la plataforma.

³XML no está predefinido, por lo que debes definir tus propias etiquetas.

Su sintaxis es muy similar al HTML. Para cada elemento de texto se tiene una etiqueta de apertura y otra de cierre, y también podemos poner etiquetas vacías. Además, el contenido o nombre de cada etiqueta lo elige el desarrollador del XML, es decir, se puede llamar a cada etiqueta con el nombre que queramos.

```
<note>
  <to>Alice</to>
  <from>Bob</from>
  <heading>Recordatorio</heading>
  <body>No te olvides de hacer la memoria</body>
</note>
```

Figura 3.2: Ejemplo código XML.

Notación de objeto JavaScript

La notación de objeto JavaScript o en inglés *JavaScript Object Notation (JSON)* [11] se trata de un formato para guardar e intercambiar información, fácil de leer y escribir para humanos, y también sencillo de analizar sintácticamente y de generar por los ordenadores.

En la sintaxis de un objeto JSON está constituida por dos elementos centrales: las claves (Cadenas de caracteres) y los valores (tipos de datos válidos por JSON⁴). Un objeto JSON es una colección desordenada [11] de cero o más pares clave-valor comenzando y terminando con llaves «{}».

```
{
  "to": "Alice",
  "from": "Bob",
  "heading": "Recordatorio",
  "body": "No te olvides de hacer la memoria"
}
```

Figura 3.3: Ejemplo código JSON.

Solución elegida

Finalmente, para escoger que lenguaje de marcado se va a emplear durante la estructuración de los mensajes entre nodos, se ha basado en el rendimiento a la hora de «serializar»⁵ datos. Según el artículo publicado por Maxim Novak [13], JSON tiene mejor rendimiento tanto a la hora de estructurar como serializar grandes y pequeñas cantidades de datos. Por lo tanto, se elige el formato JSON como estructura para el envío de mensajes entre nodos.

3.4.4. Disponibilidad de los clientes para el envío de mensajes

El mayor problema con el que tiene que lidiar una aplicación de mensajería descentralizada es la disponibilidad de los clientes, especialmente a la hora de enviar y recibir mensajes. Este problema surge porque los usuarios no dejan sus ordenadores personales

⁴Cadena de caracteres, números, objetos JSON, arrays, booleanos y null

⁵Conversión de estructuras en secuencias de bytes para su posterior envío o guardado.

encendidos todo momento. Esto es debido a que hacerlo supondría un despilfarro tanto en horas de vida del ordenador como energético. Es por ello que para combatir este problema se han propuesto las siguientes soluciones:

Uso de colas de mensajería

La cola de mensajes o en inglés *message queue*, es una forma de comunicación asíncrona de servicio a servicio que se usa en arquitecturas de microservicios⁶ y sin servidor. Existen dos clases principales de colas: no persistentes y persistentes.

- **No persistentes.** Las colas no persistentes exigen que el receptor esté activo para transmitir el mensaje.
- **Persistentes.** En las colas persistentes los mensajes se mantienen en *buffers*, por lo que el receptor no tiene por qué estar activo cuando se envía el mensaje. Además, esta clase de colas puede estar basada en el uso o no de gestores.
 - **Con gestores** o (*Broker-based*). Los sistemas de mensajería basados en gestores⁷, permiten almacenar mensajes en servidores concretos proporcionando cálculo de datos, enrutamiento, traducción de mensajes, persistencia y entrega a todos sus destinatarios.
 - **Sin gestores** o (*Brokerless*). Los sistemas de mensajería que no están basados en gestores, se basan en el almacenamiento, normalmente en memoria principal, de mensajes tanto del emisor como del receptor.

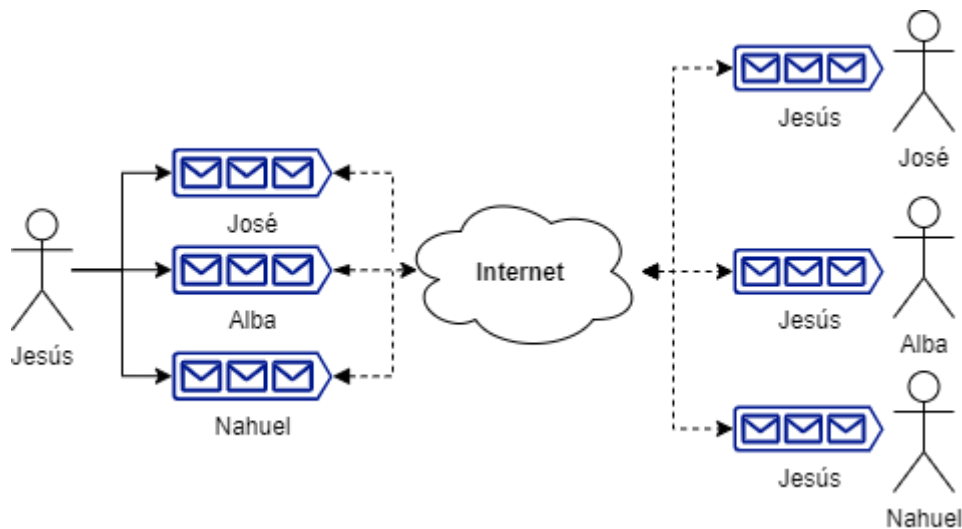


Figura 3.4: Colas de mensajería sin gestor.

Finalmente, valorando las posibles soluciones que aporta el uso de colas de mensajería, la que mejor se adapta al desarrollo de esta aplicación es el sistema de colas persistentes sin gestores, ya que este tiene la capacidad de enviar mensajes a un usuario desconectado sin necesidad de un servidor intermedio.

Almacenamiento temporal y posterior envío de mensajes

Esta solución para la transmisión se divide en dos fases.

⁶Elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas.

⁷<https://www.tibco.com/reference-center/what-is-a-message-broker>

1. **Comprobación de la disponibilidad del usuario.** Cuando un usuario proceda a enviar mensajes a un contacto, se comprueba su disponibilidad. Si el usuario se encuentra disponible, el mensaje será enviado directamente. Además, si existen mensajes sin enviar, la aplicación procederá a enviar todos los mensajes que no han sido enviados, junto con el mensaje que se desea enviar a dicho contacto. En caso contrario, el mensaje será almacenado para su posterior envío cuando el usuario destino se encuentre conectado.

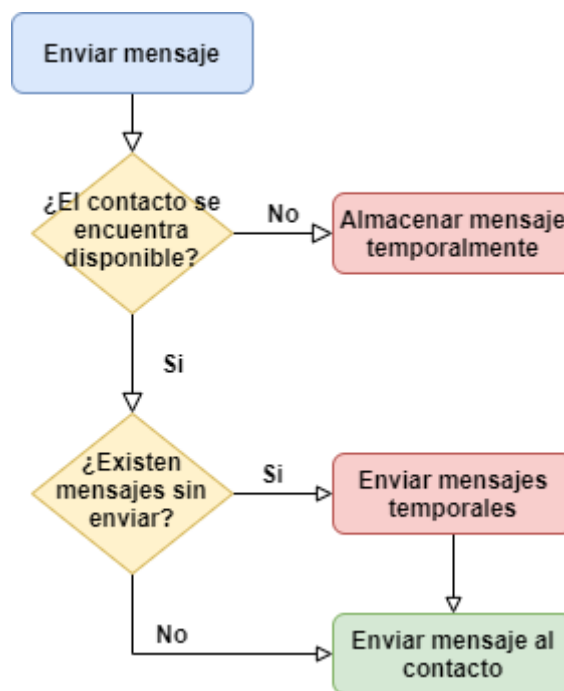


Figura 3.5: Fase 1. Transmisión de mensajes.

2. **Envío posterior de mensajes** Después de un intervalo de tiempo determinado, la aplicación activará una rutina para enviar los mensajes temporalmente almacenados a cada uno de los contactos. Para ello, seguirá los siguientes pasos:
 - a) Se procederá a comprobar entre todos los contactos almacenados cuáles tienen al menos un mensaje temporal pendiente de enviar.
 - b) Por cada uno de los contactos con mensajes pendientes se comprueba su disponibilidad.
 - c) Si el contacto se encuentra disponible se envían y eliminan los mensajes almacenados del usuario. En caso contrario, seguirán estando almacenados.

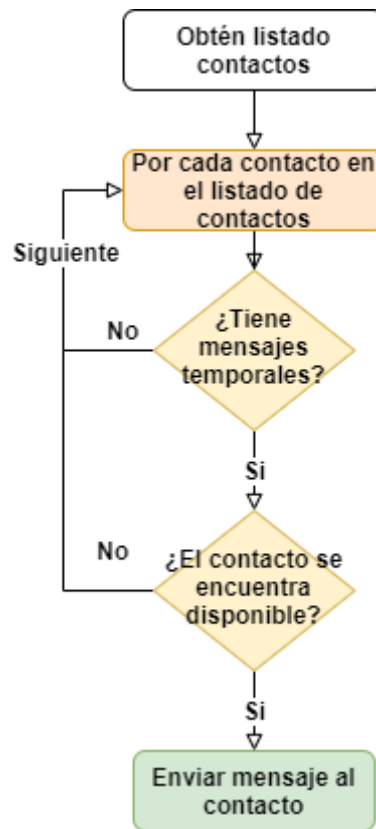


Figura 3.6: Fase 2. Reenvío de mensajes.

Solución escogida

Por un lado, la solución empleando **colas de mensajería** podría ser una buena opción, ya que haría más sencilla la implementación, por lo que todo el trabajo de envío y recepción de mensajes quedaría a cargo de las librerías de colas de mensajería *Brokerless* como **JeroMQ**⁸ y **ZMQ**⁹.

Sin embargo, puesto que estas colas se encuentran alojadas en memoria principal, si un usuario quiere hablar con n contactos, la aplicación tendrá que cargar en memoria n colas de mensajería. Esto aumentará el **consumo de memoria RAM**¹⁰ hasta que el usuario decida cerrar la aplicación o todos los mensajes hayan sido enviados a los n contactos. Además, el hecho de que el usuario cerrase la aplicación, supondría obtener todos y cada uno de los mensajes que se encuentran en memoria principal y posteriormente guardarlos en disco. Además, cada vez que el usuario inicie la aplicación, será necesario generar nuevas colas de mensajería para los mensajes que no pudieron ser enviados anteriormente.

Finalmente, el uso de colas de mensajería no es una solución tan acertada como uno podría pensar en un principio, sobre todo si se plantea hacer uso de esta aplicación en equipos informáticos con poca memoria o poco poder de cómputo.

Por consiguiente, si se comprueba la disponibilidad de un contacto antes de enviar un mensaje y en base a ello se almacena o se envía el mensaje. Al iniciar la aplicación, solo tendría que existir una **única rutina** en memoria principal, encargada de enviar todos los mensajes temporales. De este modo, se consume menor cantidad de recursos. Siendo

⁸<https://github.com/zeromq/zeromq>

⁹<http://zguide.zeromq.org/page:all>

¹⁰Random Access Memory

una mejor solución que la anteriormente planteada y por tanto será la que se implantará durante el desarrollo del código de la aplicación.

3.4.5. Seguridad

Desde el punto de vista de la seguridad, un mensaje se ha transmitido de forma segura si durante su transmisión se han cumplido las siguientes propiedades:

- **Autenticación.** Tanto el emisor como el receptor están seguros de que el mensaje ha sido enviado o recibido por quien dice ser.
- **Confidencialidad.** Solamente el emisor y el receptor conocen el contenido del mensaje.
- **Integridad.** El mensaje no ha sido modificado durante su transporte.

Para garantizar las propiedades anteriormente mencionadas se proponen diferentes soluciones:

Autenticación

Para poder asegurar la autenticación en la transmisión de mensajes existen muchos métodos, protocolos y sistemas en la actualidad. Sin embargo, la solución se basará en uno de los siguientes tipos de protocolos de autenticación[22], porque fueron diseñados para validar la autenticidad de los clientes a través en los protocolos punto a punto¹¹:

- **PAP - Password Authentication Protocol.** El usuario envía las credenciales en **texto claro** al servidor. El servidor compara las credenciales con las que el tiene almacenadas, verificando si el usuario es quien dice ser.

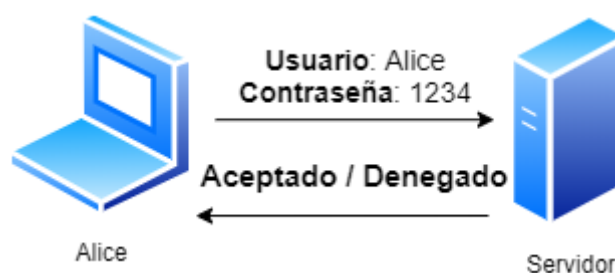


Figura 3.7: Password Authentication Protocol.

- **CHAP - Challenge-handshake authentication protocol.** El servidor envía una cadena de caracteres aleatoria. El cliente utiliza la cadena de caracteres y la contraseña del usuario como parámetros de una función *hash* MD5. El resultado de la función *hash* junto con el **nombre del usuario** son enviados al servidor en texto claro. El servidor compara el resultado recibido con el calculado a partir de la contraseña que el tiene almacenada, comprobando si la contraseña es correcta y el usuario es quien dice ser.

¹¹Utilizados para establecer una conexión directa entre dos nodos de una red.

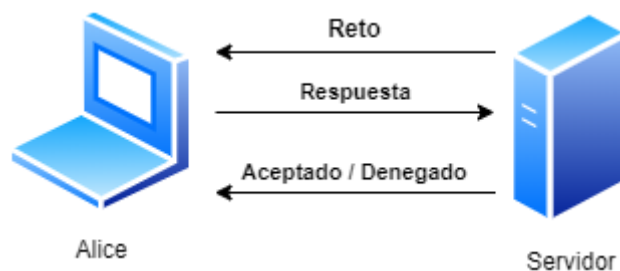


Figura 3.8: Challenge-handshake authentication protocol.

Los protocolos anteriormente mencionados se basan en el conocimiento de un secreto conocido entre los dos puntos, este secreto suele ser la contraseña del usuario a autenticar. Sin embargo PAP tiene un par de desventajas con respecto a CHAP. Con PAP las credenciales son enviadas en texto claro, permitiendo que un atacante pueda obtenerlas y pueda realizar ataques de suplantación. Además, PAP permite que un atacante con credenciales robadas pueda autenticarse tantas veces como quiera frente a cualquier servidor. Con CHAP, debido a la generación de retos, el atacante solamente podrá autenticarse una vez con la solución del reto robada del cliente.

No obstante, CHAP no es perfecto, pues se basa en el uso de la función *hash* MD5, haciéndolo inseguro. **MD5 es vulnerable** frente a ataques de búsqueda colisiones [23]. Por tanto, para evitar esta vulnerabilidad se reemplaza la función MD5 por una función más actual, como es el caso de SHA-256. Esta función se considera segura porque todavía no se han encontrado colisiones debido al tamaño de su salida (256 bits). En otras palabras, la probabilidad de encontrar una colisión es de 1 entre $1,1579209 \times 10^{77}$, este número es comparable al número de átomos en el universo, entre 10^{78} y 10^{82} [24].

El empleo de estos protocolos requieren del uso de un servidor central, encargado de autenticar a todos los usuarios de la red. Como se quiere evitar lo máximo posible el uso de un servidor central para el desarrollo de esta aplicación, se plantea la solución de que cada cliente emplee el protocolo CHAP para autenticar a los contactos con los que el usuario desea hablar. Para ello, los clientes tendrán que almacenar un secreto precompartido entre usuario y el contacto. Por ejemplo, el usuario y el contacto han establecido, a través del cara a cara, la frase «2000 leguas de viaje submarino» como su secreto para autenticarse mutuamente.

Finalmente, para proporcionar autenticidad entre los miembros de la aplicación, se empleará el protocolo CHAP empleando SHA-256 como función *hash* y se utilizará como «secreto» una cadena de caracteres precompartida entre el usuarios.

Confidencialidad

Garantizar que los mensaje enviados jamás sean interceptados es una tarea imposible debido a que el modelo de «interconexión de sistemas abiertos» (OSI) no es perfecto pues existen diferentes métodos de obtención de información como: la captura de paquetes de datos mediante «sniffers»[20], la captura de paquetes a través de medios inalámbricos [18] y los ataques de hombre en el medio [19]. Por lo tanto, es necesario cifrar la información enviada, dificultando al confidente obtener el mensaje oculto a pesar de haberlo capturado. Además, dicha dificultad aumenta si se proporciona *Perfect Forward Secrecy*, como se comenta en la sección 2.2 Propuesta.

Para empezar, *Perfect Forward Secrecy* proporciona una mejor protección que otros métodos, debido al hecho de generar una única clave por cada transacción entre pares. Para

ello, se necesitan un protocolo de establecimiento de claves efímeras para medios inseguros y un algoritmo de cifrado simétrico.

Por un lado, para los protocolos de establecimiento de claves efímeras se han propuesto los más usados actualmente.

- Diffie-Hellman (DH)** [27] emplea la exponenciación modular y la generación de números pseudoprimos¹² para la elaboración de las claves públicas (A y B) y la clave compartida (K), siendo a y b las claves privadas (números aleatorios dentro del intervalo $[1, p - 1]$) y g un generador (raíz primitiva¹³). Diffie-Hellman es un algoritmo que basa su seguridad en el problema del logaritmo discreto.

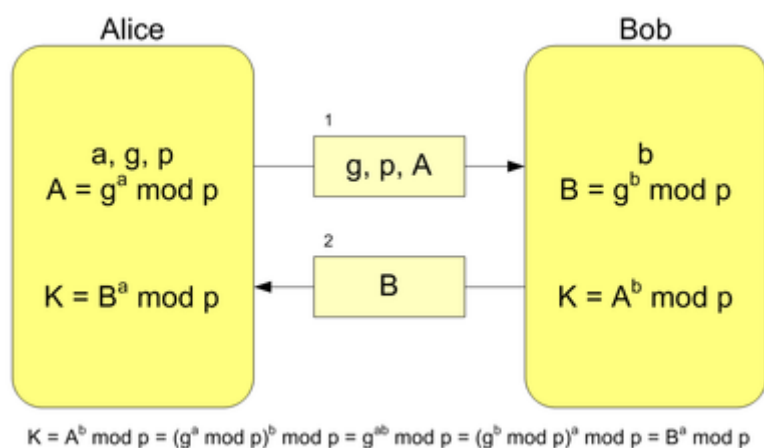


Figura 3.9: Diffie-Hellman Key Exchange.

- Elliptic-Curve Diffie-Hellman (ECDH)** es una variante del protocolo Diffie-Hellman que utiliza la criptografía de curvas elípticas para la generación de un par de claves. La clave privada es un número generado aleatoriamente d dentro del intervalo $[1, n - 1]$, siendo n un número primo, y la clave pública es el punto de coordenadas resultante de multiplicar d veces p , siendo p un número conocido por ambas partes. Al igual que en Diffie-Hellman, la clave compartida es el resultado de multiplicar la clave privada de A por la clave pública de B y viceversa. Finalmente, al igual que DH, la seguridad de ECDH se basa en el problema del logaritmo discreto, sin embargo enfocado a las curvas elípticas.

¹²Se dice que n es pseudoprime si n divide a $b^{n-1} - 1$

¹³Decidamos que a es raíz primitiva si $\forall b \in \mathbb{Z}_n^*$ existe $k \in \mathbb{Z}$ tal que $a^k \equiv b \pmod n$

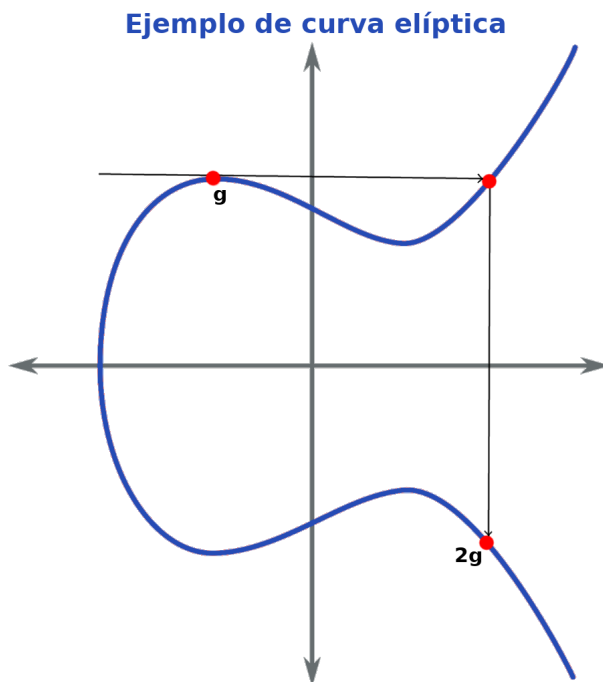


Figura 3.10: Aproximación a la generación de claves públicas con curvas elípticas.

Se escoge ECDH como solución, pues con un tamaño menor de clave, proporciona la misma seguridad que otros algoritmos con tamaños de clave mayores, como Diffie-Hellman o RSA [26]. Además, la generación de estas claves son más rápidas, porque requieren de menor poder de cómputo y por ende se necesita un menor tiempo de espera para su generación.

Además, para demostrar la eficiencia de ECDH frente a DH, se ha desarrollado una prueba que consiste en comprobar el tiempo requerido en el envío de un mismo mensaje entre dos nodos, siendo cifrado con AES cuya clave empleada es el resultado de los protocolos ECDH y DH.

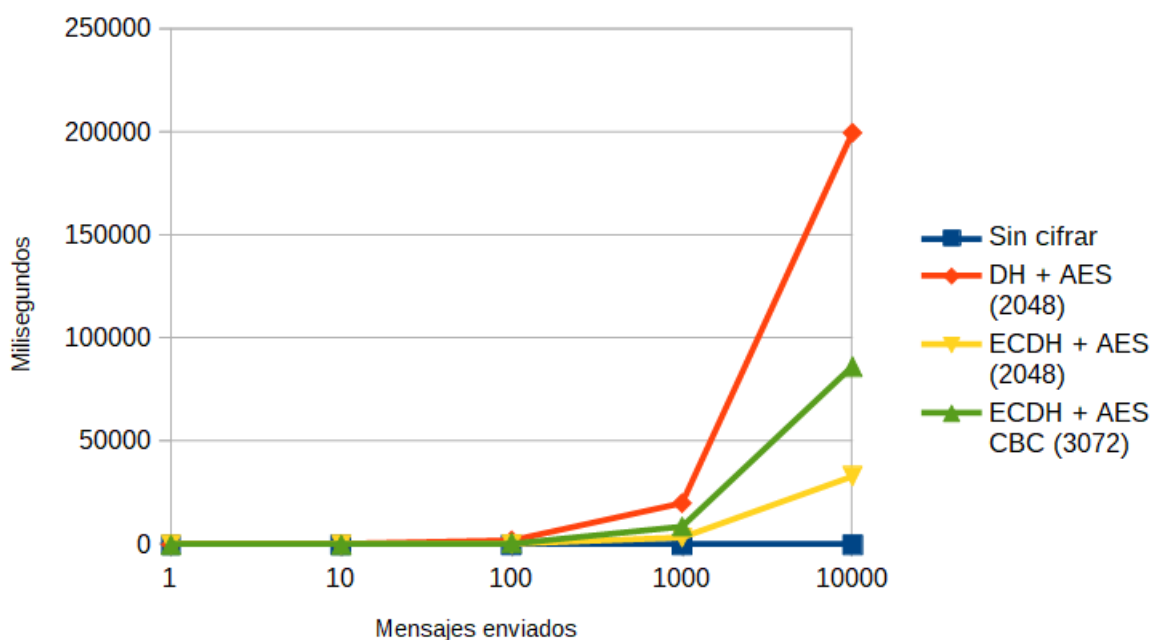


Figura 3.11: Gráfica de tiempos de ECDH y DH.

En la siguiente tabla se pueden apreciar los milisegundos requeridos para el envío de 1, 10, 100, 1000 y 10000 mensajes respectivamente, en cada uno de los diferentes sistemas de cifra.

(Milisegundos)	1 mensaje	10 mensajes	100 mensajes	1000 mensajes	10000 mensajes
Sin cifrar	0	0	2,1	24,4	248,7
DH + AES (2048)	22,5	210,1	2057,8	20157,6	199922,3
ECDH + AES (2048)	15,6	35,7	337,9	3497,5	33149,5
ECDH + AES CBC (3072)	21,0	89,5	894,2	8746,8	86552,0

Tabla 3.1: Tabla de tiempos de ECDH y DH.

Como se puede ver en las imágenes anteriores ECDH sigue siendo más eficiente que DH, a pesar de haber generado claves equivalentes a claves DH de 3072 bits, según el *Internet Engineering Task Force (ietf)* o en castellano «Grupo de Trabajo de Ingeniería de Internet» [21].

Por otro lado, se propone como algoritmo de cifrado de clave simétrica *Advanced Encryption Standard (AES)* [30], también conocido como Rijndael (pronunciado «Rain Doll» en inglés). Es un algoritmo de cifrado simétrico que emplea la misma clave tanto para el cifrado como para el descifrado de un mensaje, residiendo su seguridad en la clave empleada. Es el sucesor de *Data Encryption Standard (DES)* [28] tras ser declarado inseguro [29] en el año 2001. A pesar de ser un algoritmo de cifrado por bloques como DES. AES difiere de DES en el empleo de sustitución y permutación de celdas en una matriz, en vez de la utilización de las redes de «Feistel» empleadas en DES [31]. Además, es necesario incidir en que AES posee diferentes modos de cifra, de entre todos ellos se destacan los siguientes[32]:

- El modo *Electronic CodeBook (ECB)* es el más simple de todos. El texto plano se divide en bloques de un tamaño fijo, añadiendo relleno hasta llenar el bloque. Después, cada bloque es cifrado con la misma clave y algoritmo, por lo que si se cifra el mismo texto varias veces, se obtendrá siempre el mismo criptograma.

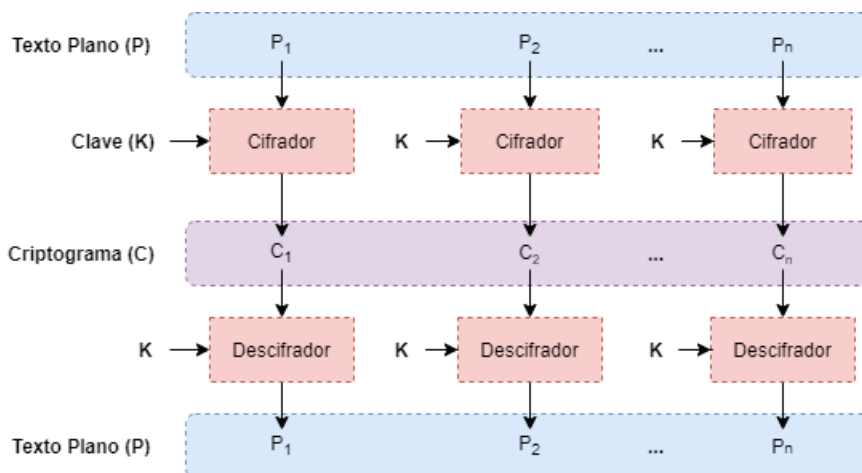


Figura 3.12: Algoritmo de cifrado AES modo ECB.

- El modo *Cipher Block Chaining* (**CBC**) es un cifrador de bloques al igual que ECB. Antes de cifrar un bloque se le aplica una operación XOR con el bloque previo ya cifrado. Además, al primer bloque se realiza la operación XOR con un vector de inicialización (**IV**)¹⁴ haciendo que el cifrado de cada mensaje sea único.

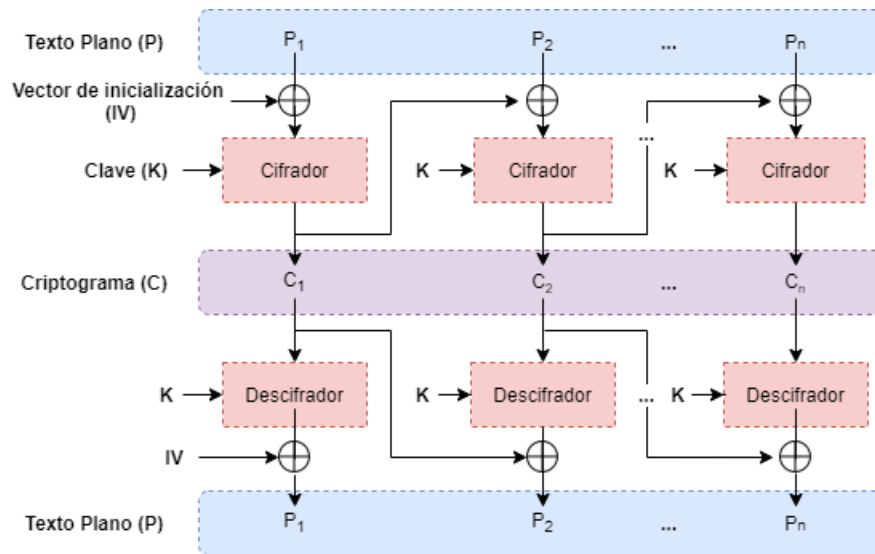


Figura 3.13: Algoritmo de cifrado AES modo CBC.

- El modo *Cipher FeedBack* (**CFB**) es un cifrador de bloques que cifra como si de un cifrador de flujo¹⁵ se tratase. Primero se cifra el vector de inicialización, después se realiza la operación XOR con el texto plano, obteniendo un criptograma. Posteriormente, el criptograma se utiliza para realizar la operación XOR con el siguiente bloque en texto plano y su posterior cifrado.

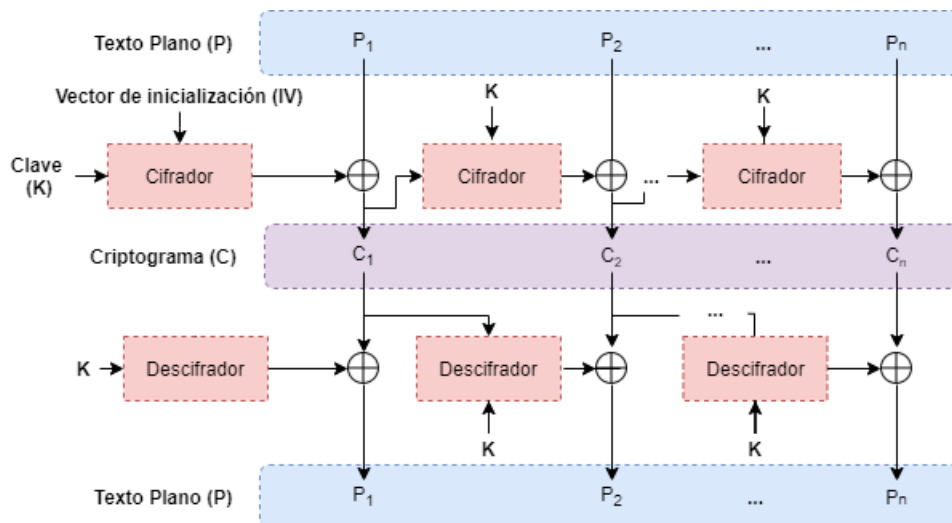


Figura 3.14: Algoritmo de cifrado AES modo CFB.

- En el modo *Output FeedBack* (**OFB**) al igual que el modo CFB es empleado como si un cifrador de flujo se tratase. En este modo, al igual que en el modo CFB, se cifra el IV, para emplear el resultado como parámetro para la operación XOR junto con

¹⁴Bloque con una longitud fija de bits generados aleatoriamente

¹⁵Un cifrador de flujo es un cifrado de clave simétrica donde el texto plano se combina con un flujo de dígitos pseudoaleatorios

el texto plano. Sin embargo, OFB difiere de CFB, en que la clave para el siguiente bloque es el resultado del cifrado, en vez del resultado de la operación XOR.

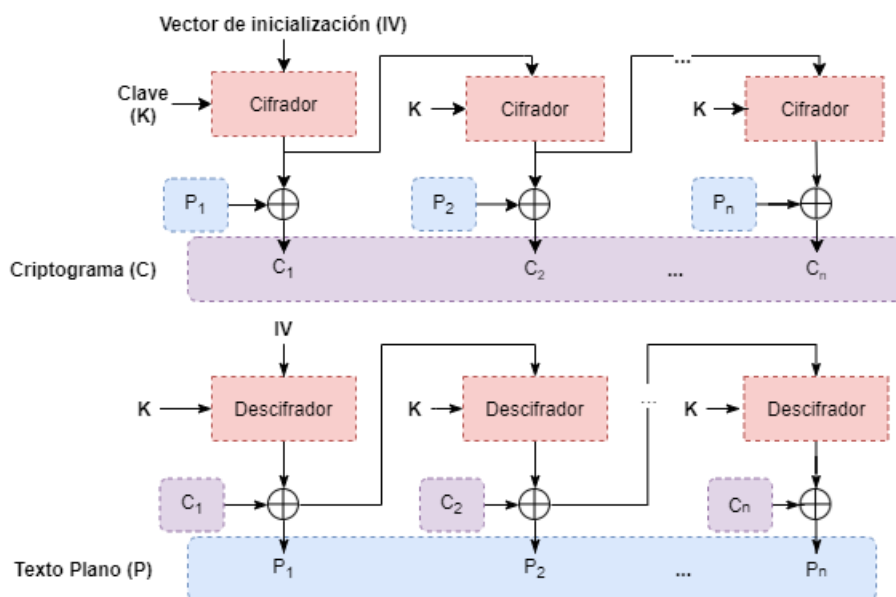


Figura 3.15: Algoritmo de cifrado AES modo OFB.

- En el modo *Counter* (CTR) es un cifrador de bloque empleado como un cifrador de flujo. En vez de emplear un vector de inicialización, utiliza un contador con longitud equivalente al tamaño del bloque. El contador se cifra con la clave simétrica, el resultado junto con el texto plano es empleado como parámetro para la operación XOR obteniendo el criptograma.

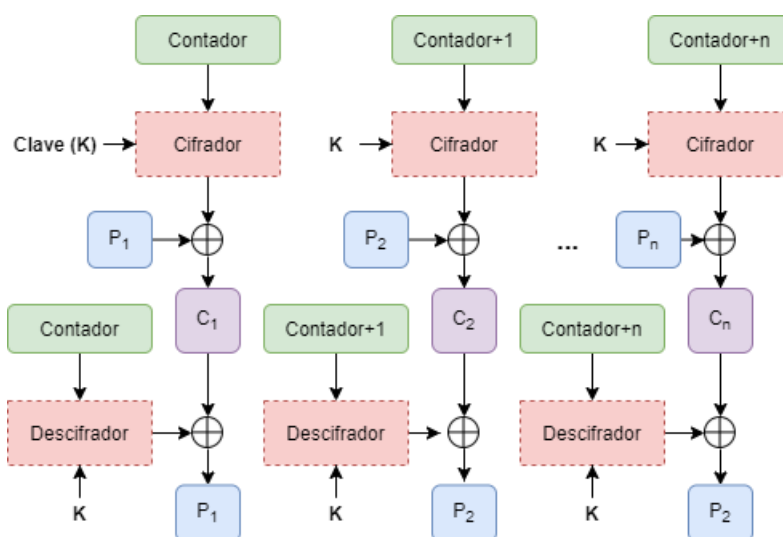


Figura 3.16: Algoritmo de cifrado AES modo CTR.

Entre todos los modos mencionados anteriormente, se ha decantado por el modo *Cipher Block Chaining*, puesto que CBC es el único que al producirse una modificación de cualquier bit en el criptograma, supondría que el mensaje descifrado fuese ilegible y por tanto, se sabría que el mensaje ha sido modificado.

Finalmente, para la confidencialidad de los mensajes se empleará *Perfect Forward Secrecy* basada en el uso de ECDH como protocolo de establecimiento de claves efímeras y en el empleo de AES usando el modo CBC para el cifrado simétrico de los mensajes.

Integridad

La solución para comprobar la integridad de los mensajes se fundamenta en el empleo de códigos de autenticación de mensajes en clave-hash o en inglés *keyed-Hash Message Authentication Code (HMAC)*. HMAC es un tipo de código de autenticación de mensajes, en inglés *Message Authentication Code (MAC)*, que implica el uso de funciones *hash*. MAC es un resumen criptográfico que permite verificar simultáneamente la integridad de los datos y la autenticación de un mensaje a través de una información resultante a partir de un mensaje y una clave precompartida entre el remitente y el destinatario. Por lo tanto, si un atacante desea modificar un mensaje, tiene que obtener previamente el secreto precompartido entre los usuarios que forman la conversación, dificultando enormemente la modificación de los mensajes.

Paralelamente, como la estructura de los mensajes se basa en el uso de objetos JSON, permitiendo la aleatorización de los campos que componen un mensaje. Es decir, si creamos diferentes objetos JSON cada uno con las mismas claves: «texto del mensaje», «fecha del mensaje» y «hash del mensaje», pero con diferentes valores; el campo «texto del mensaje» podría encontrarse antes o después del par «fecha del mensaje» o del par «hash del mensaje». Además, al estar todos los mensajes cifrados con claves diferentes, dificulta enormemente la posibilidad de que un atacante modifique un mensaje sin que el receptor pueda detectarlo. Puesto que en ningún momento, el atacante será consciente ni de la posición ni de la longitud que ocupan los mismos. No obstante, esta aleatoriedad solo funciona cuando se envían mensajes diferentes. En otras palabras, si un usuario envía varios mensajes iguales, los componentes del mensaje se encontrarán en la misma posición, facilitando al atacante la modificación de dichos mensajes.

Por consiguiente, si a un mensaje le añadimos varios pares clave-valor aleatorios en el objeto JSON, producirá mayor aleatoriedad en el posicionamiento de los campos que componen un mensaje a pesar de haber escrito el mismo mensaje.

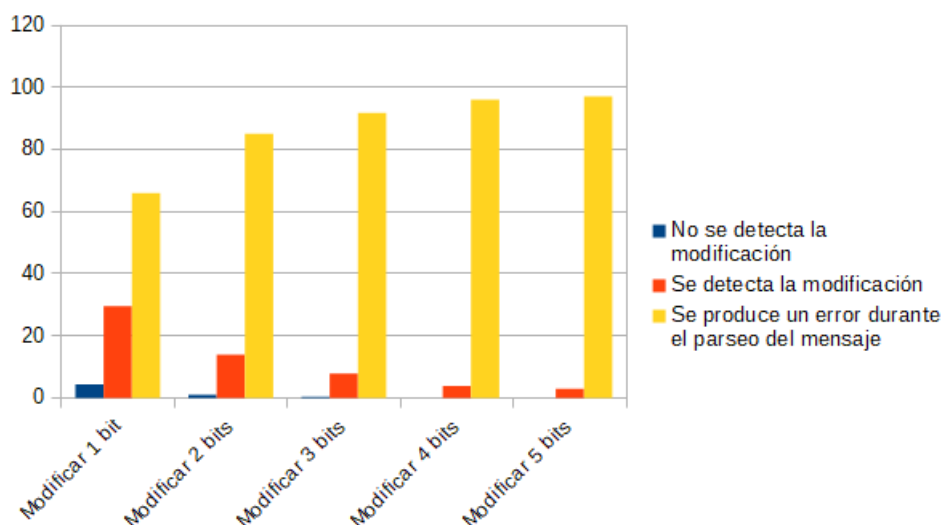


Figura 3.17: Gráfica de la comprobación de integridad de los mensajes.

De cada 100 mensajes	No se detecta la modificación	Se detecta la modificación	Se produce un error durante el parseo del mensaje
Modificar 1 bit	4,4	29,6	66
Modificar 2 bits	1	13,9	85,1
Modificar 3 bits	0,4	7,8	91,8
Modificar 4 bits	0	3,9	96,1
Modificar 5 bits	0	2,9	97,1

Tabla 3.2: Tabla de la comprobación de integridad de los mensajes.

Como se puede ver en las figuras anteriores. Al principio no se detectan algunas modificaciones. Esto puede ser debido a que el bit que ha sido modificado pertenece a los pares de relleno que contiene el mensaje, dejando la estructura del mensaje intacta. Además, como también se puede apreciar en la mayoría de los casos, se produce un error durante el análisis del contenido del mensaje, impidiendo que el mensaje pueda ser leído por el receptor.

Por consiguiente, se implantará el uso de HMAC como solución en el desarrollo de la aplicación, porque aparte de detectar si un mensaje ha sido enviado correctamente, en el peor de los casos, un atacante solamente será capaz de evitar que los mensajes puedan ser procesados por la aplicación.

3.4.6. Solución final

La solución resultante de todos los problemas que aparecieron durante el análisis de la transmisión de mensajes será: el uso del protocolo **TCP**, para asegurar la correcta recepción de los paquetes enviados; la implementación del método de **Almacenamiento temporal y posterior envío de mensajes** para poder enviar los mensajes una vez el contacto se encuentre disponible, el sistema *Diffie-Hellman* y *AES-256* para la confidencialidad de los mensajes, el seguimiento del protocolo **CHAP** para asegurar la autenticidad del contacto y el de HMAC para asegurar tanto la integridad como la autenticación de los mensajes.

3.5 Permitir la recepción de mensajes de desconocidos

Puede darse el caso que un usuario *A* le interese recibir mensajes de otro usuario *B* completamente ajenos a él. Sin embargo, como se ha visto en la sección 3.4.5 *Integridad*, para que dos usuarios puedan autenticarse mutuamente es necesario un secreto precompartido, el cual el usuario *B* no tiene.

La solución este problema consiste en que el cliente del usuario *B*, que a partir de ahora será mencionado directamente como *B*, envía el identificador de su usuario, para verificar si el usuario *A* lo tiene agregado. Si el identificador no se encuentra registrado en la base de datos y el usuario *A* permite mensajes de desconocidos, *B* enviará al usuario *A* el secreto con el que usuario *A* fue agregado en el cliente del usuario *B*. Este secreto es empleado por el cliente del usuario *A* para agregar un nuevo contacto y así poder autenticar todos los mensajes recibidos del usuario *B*.

No obstante, cabe expresar que si un ciberdelincuente captura y descifra el mensaje que contiene el secreto transmitido, este ciberdelincuente puede hacerse pasar por cualquiera de ambos usuarios *A* o *B*. Por lo tanto, se recomienda que se haga un uso respon-

sable sobre esta característica de la aplicación. Además, se recomienda acordar un nuevo secreto compartido, una vez ambos usuarios hayan entablado una conversación.

3.6 Obtención de la ubicación de los usuarios

En la mayoría de las aplicaciones de mensajería actuales, los mensajes son enviados a uno o varios servidores centrales, encargados de hacer llegar el mensaje a su destinatario. Sin embargo, antes de enviar cualquier mensaje, el cliente necesita saber dónde se encuentra dicho servidor, para ello se emplean las direcciones IP. Las direcciones IP pueden ser estáticas o dinámicas en función de si varían o no con el tiempo. En la gran mayoría de los casos, la IP asignada es dinámica. Esto es un problema, porque en el caso de esta aplicación, la dirección IP y el puerto asociado a cada contacto tienen que ser actualizados cada vez que a un contacto se le haya asignado una nueva dirección IP o un nuevo puerto.

Con la finalidad de resolver este problema se han propuesto las siguientes soluciones.

3.6.1. Solicitud manual

Esta solución se basa en el uso de aplicaciones de mensajería externas usadas por el usuario para la obtención de la dirección IP pública [16] y del puerto a la escucha del contacto. Para clarificar en que consiste esta solución, se presenta el siguiente caso de uso:

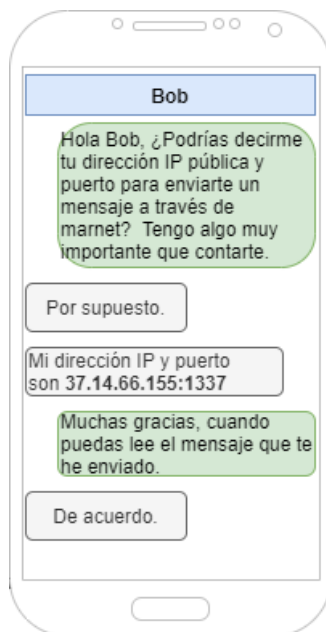


Figura 3.18: Pedir una dirección IP y un puerto mediante otra aplicación.

1. Alice quiere enviarle un mensaje a Bob.
2. Alice le pregunta a Bob a través de una aplicación de mensajera, por ejemplo, Telegram. Si le podría enviar su dirección IP pública y el puerto al que se encuentra a la escucha para entablar una conversación.
3. Bob envía su dirección IP y puerto a Alice.
4. Alice introduce la IP y el puerto de Bob en la aplicación y procede a entablar una conversación con Bob.

Como el lector podrá entender esto supone un gran esfuerzo, pues si se quiere enviar un mensaje a n contactos se tiene que solicitar la dirección IP pública y el puerto a la escucha n veces. Además, si el usuario no se encuentra conectado y se almacena el mensaje, cada vez que se quiera reenviar el mensaje, la aplicación tendrá que solicitar al usuario la dirección IP y el puerto del contacto, pues puede que estos valores hayan variado desde la última vez que se utilizaron. Por lo tanto, esta solución queda tachada como inviable.

3.6.2. Servidor para la resolución de identificadores

La solución que se propone es la creación de un servidor central, que permita obtener la dirección IP y el puerto de un usuario a partir de un identificador proporcionado. Es decir, si Alice desea hablar con Bob, Alice deberá preguntar al servidor de identificadores la dirección IP y el puerto correspondiente al identificador de Bob. Este identificador debe ser único, pues si en un mismo servidor alberga varios usuarios con el mismo identificador, podría suponer un vector de ataque para la realización de ataques de suplantación de identidad, resultando en una gran brecha de seguridad. Además, merece la pena indicar que esta solución es empleada en aplicaciones como **bitTorrent**, **deluge**, **μ Torrent**, etc. donde un servidor central, llamado **Tracker**, contiene toda la información necesaria para que los pares de dichas aplicaciones puedan conectarse entre sí.

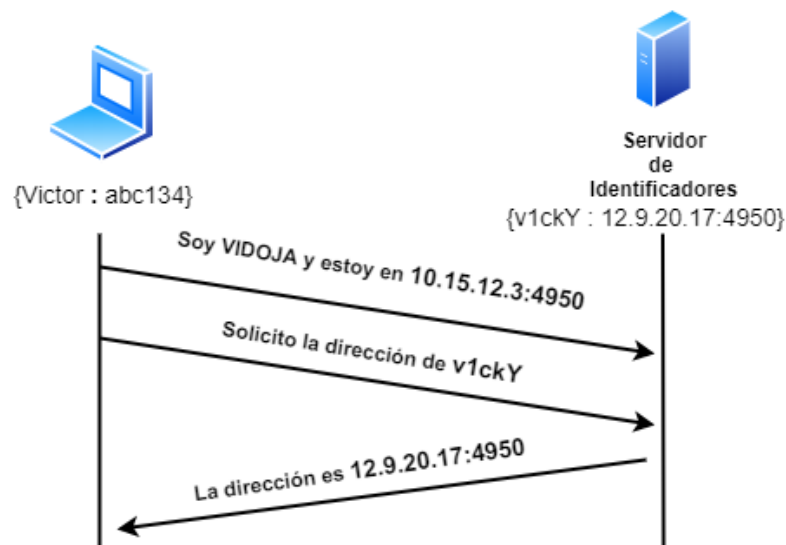


Figura 3.19: Servidor de identificadores. Solicitud de IP.

3.6.3. Solución escogida

Debido a las ventajas que supone el servidor de identificadores frente a preguntar a cada uno de los usuarios por su dirección IP y su puerto a la escucha, Se elige como solución, la creación de un servidor de identificadores.

3.6.4. Seguridad

Para asegurar las conexiones entre clientes y el servidor de identificadores, y para evitar posibles ataques de fuerza bruta (evitando que los atacantes conozcan que usuarios están registrados en el servidor); se han realizado las siguientes medidas:

Conexiones con los clientes

Para proporcionar una conexión segura entre los clientes y el servidor se han elegido las siguientes soluciones:

- **Perfect Forward Secrecy (ECDH y AES-256 [CBC])** para proporcionar confidencialidad, al igual que los clientes.
- **SHA-256** para la integridad de los mensajes.

Es necesario incidir en que como la autenticidad entre usuarios reside en los clientes, no es necesaria implementarla en el lado del servidor. Es por ello, que en vez de usar HMAC para comprobar si un mensaje ha sido modificado, se emplea directamente el uso de la función resumen SHA-256, añadiendo como parámetros los valores de los campos que componen el mensaje enviado al servidor.

Ataque de fuerza bruta

Para evitar usuarios mal intencionados conozcan los identificadores de otros usuarios conectados al servidor de identificadores se ha propuesto como solución el empleo de lista blanca y lista negra. De esta forma, el administrador del servidor podrá permitir que solo ciertas direcciones IP se conecten al servidor de identificadores o podrá bloquear solamente ciertas direcciones IP que el administrador considere sospechosos.

3.6.5. Solución final

Como solución final se desarrollará un servidor de identificadores cuyas conexiones tendrán *Perfect Forward Secrecy* para proporcionar confidencialidad entre el cliente y el servidor y proporcionarán integridad de los mensajes. Además, el servidor tendrá la capacidad de prohibir o permitir la conexión a determinados usuarios, a través del uso de una lista «blanca» o «negra».

3.7 Redirección de puertos

En secciones anteriores se comentaba que los clientes no solamente se encargan de enviar los mensajes a su destino, sino que también están encargados de recibirlos. Para ello, los clientes tienen un puerto a la escucha, esperando a que las demás aplicaciones se conecten a él y procedan a enviar los mensajes destinados al usuario. Además, también se ha comentado sobre el uso de un servidor para la localización de clientes que también necesita un puerto a la escucha, para recibir los identificadores de los usuarios. Este método es perfecto cuando tanto el servidor y los clientes pertenecen a una misma red. Sin embargo, si dos o más clientes se encuentran en diferentes redes, es necesario indicar al router que redirija los paquetes recibidos al ordenador de la red interna que los este esperando. Por consiguiente, se han propuesto diferentes soluciones para este problema.

3.7.1. Redirección manual de puertos

La redirección manual de puertos o en inglés *port forwarding* consiste en que los usuarios de la aplicación tengan que acceder al panel de administración de su router, ya sea mediante HTTP, SSH, TELNET, etc. configurándolo para que todos los paquetes que reciba en un puerto específico sea enviado a un puerto y a un equipo específico dentro de la red. La ventaja que proporciona este método, es que el usuario tiene el control absoluto de la red y puede decidir en todo momento si quiere o no recibir mensajes provenientes de otras redes ajenas a la suya. Sin embargo, la desventaja asociada a esta solución, es la necesidad de estar constantemente creando y eliminando redirecciones de puertos.

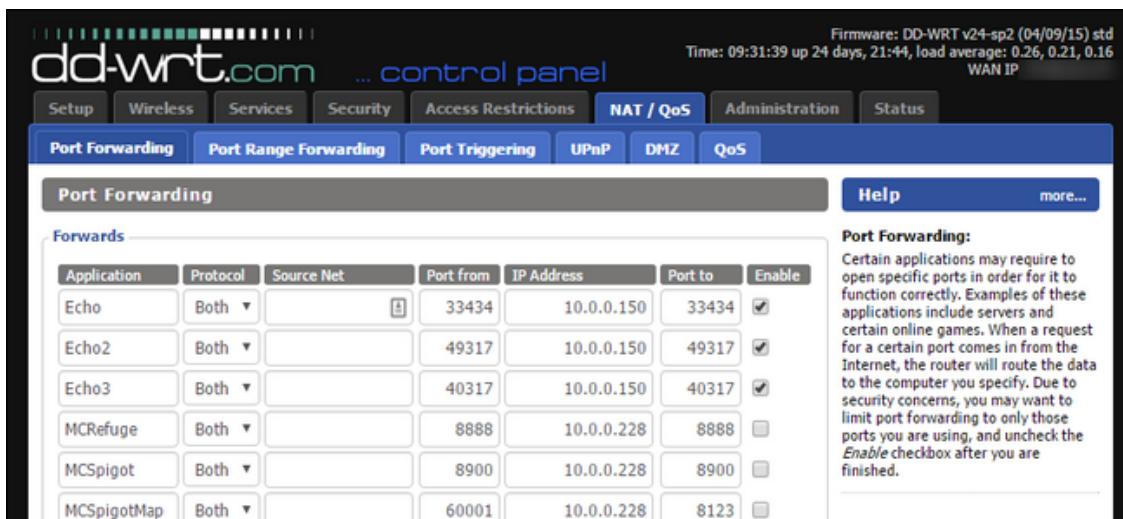


Figura 3.20: Reenvío de puertos manual.

3.7.2. UPnP

Esta solución consiste en el uso de *Universal Plug and Play* o simplemente **UPnP**. UPnP es un conjunto de protocolos de comunicación que permiten comunicarse entre sí diferentes dispositivos situados en una misma red. Uno de estos protocolos es el denominado *Internet Gateway Device Protocol* o protocolo IGD, que permite, entre otras muchas cosas, añadir o eliminar redireccionamiento de puertos. La ventaja que tiene esta solución es que el redireccionamiento se puede hacer automático, sin que el usuario de la aplicación se tenga que preocupar de dicha tarea. No obstante, puede darse el caso que en algunos rúters antiguos UPnP no este implementado, por lo que puede que no sea una solución apta para todos los usuarios.

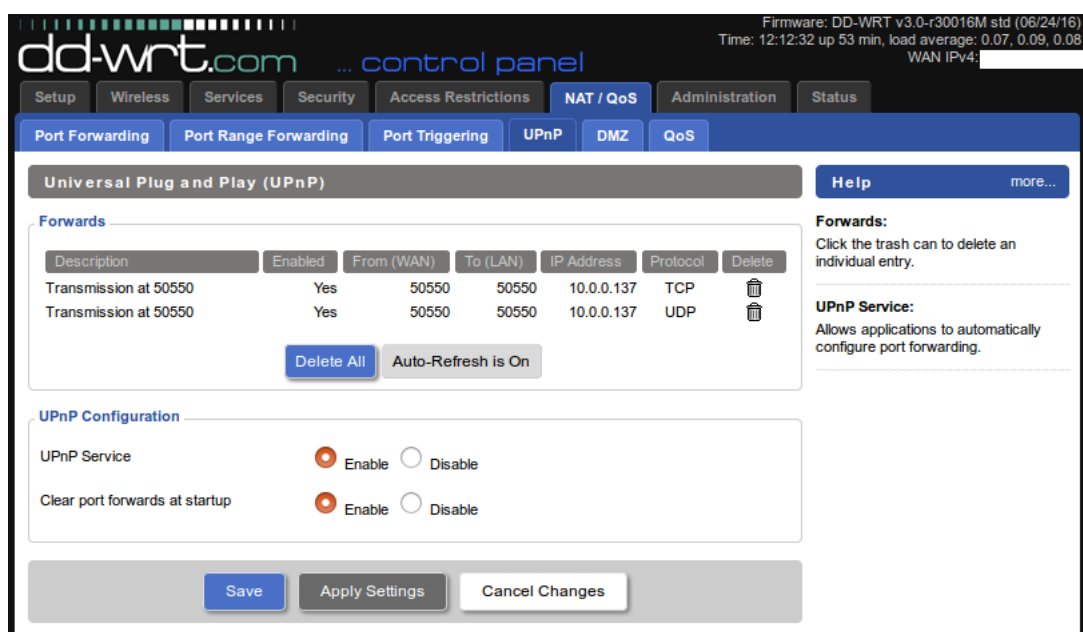


Figura 3.21: UPnP rúter dd-wrt.

3.7.3. DMZ

Esta solución consiste en el uso de la zona desmilitarizada o en inglés *demilitarized zone* (DMZ), que se puede encontrar en algunos rúters domésticos. Una DMZ es una red aislada que se encuentra entre la red interna y la red externa (normalmente Internet). La DMZ permite conexiones procedentes tanto de Internet, como de la red local. Sin embargo, no se permiten conexiones desde la DMZ a la red local.

Para poder llevar a cabo esta solución, el usuario deberá mantener un equipo en la DMZ con la aplicación instalada en él. Las ventajas de esta solución es que el usuario no tendrá que lidiar ni con el uso de UPnP ni con la redirección de puertos. Además, si el equipo resulta infectado por alguna vulnerabilidad que pueda existir en el equipo, el resto de la red no se vería comprometida. Sin embargo, la desventaja del uso de la DMZ es que el usuario tendrá que fortificar su equipo para evitar que ciberdelincuentes puedan atacar y comprometer el ordenador, pues estará expuesto a Internet frente a una gran variedad de ataques realizados diariamente.

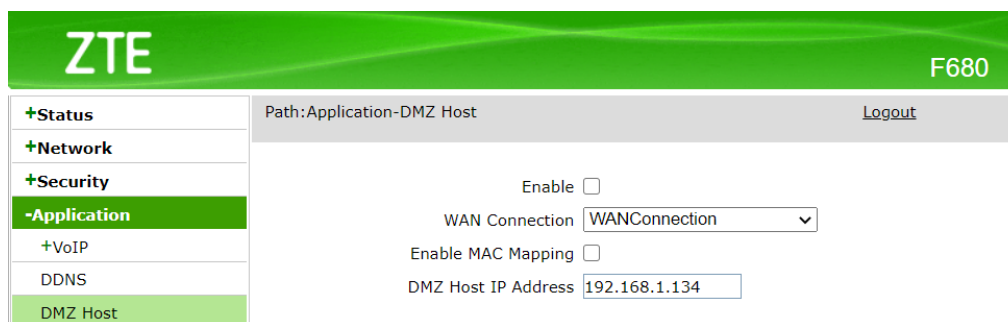


Figura 3.22: DMZ rúter ZTE.

3.7.4. Solución escogida

Después de mencionar todas las soluciones para el redireccionamiento de puertos, se consideran que todas son igual de válidas y se deja al usuario final la elección de la solución. Por consiguiente, se implementará UPnP en la aplicación y se cede al usuario final si desea emplearla o no.

3.8 Almacenamiento de información

El almacenamiento de datos es un apartado fundamental a la hora de desarrollar un programa que necesita persistencia de los datos, tanto en el lado del cliente como en el servidor.

3.8.1. Aplicación

Para poder almacenar toda la información generada por el usuario en el equipo, se plantean las siguientes soluciones:

Árbol de ficheros

Esta solución se basa en el almacenamiento de la información en diferentes ficheros, siguiendo la estructura de un árbol de ficheros, el cual contendría cuatro carpetas prin-

cipales, cada una con propósito diferente: almacenamiento de identificadores de usuario bloqueados, almacenamiento de archivos compartidos con el usuario, configuración de usuarios registrados e información sobre los contactos y conversaciones de cada contacto.

Dentro de las carpetas «Descargas» y «Conversaciones» existiría una carpeta extra con el nombre del usuario registrado en la aplicación. En el caso de la carpeta «Descargas» se almacenarían todos los archivos compartidos con el usuario. En cambio, en la carpeta «Conversaciones» se almacenarían una única carpeta por cada contacto agregado, donde se guardaría la información del contacto, los mensajes de la conversación con el usuario y los posibles mensajes temporales que todavía no han podido ser enviados al contacto.

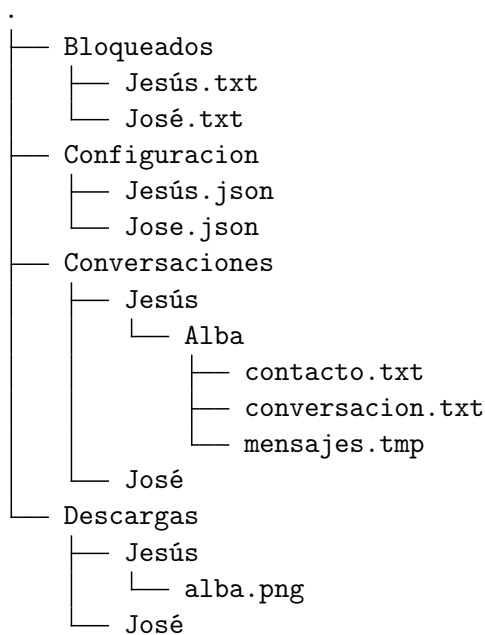


Figura 3.23: Sistema de almacenamiento. Árbol de ficheros

Base de datos

La solución del uso de una base de datos se basa en almacenar toda la información generada por el usuario, mencionada previamente, en diferentes tablas de la base de datos.

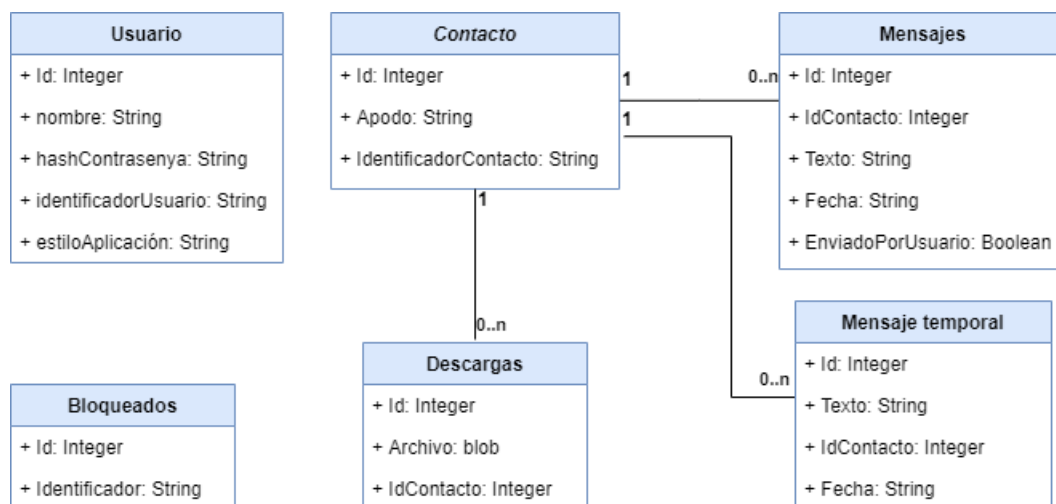


Figura 3.24: Sistema de almacenamiento. Base de datos

Como se puede ver en la imagen anterior, en vez de existir cuatro carpetas principales, existirían cinco tablas en la base de datos, resultado de dividir la conversación con el contacto y la información del contacto en dos tablas diferentes y de mover los mensajes temporales de la carpeta del contacto a una tabla diferente. Además, cada usuario tendrá su propio fichero de base de datos, de esta forma se asegura que ningún usuario pueda acceder a la información de cualquier otro usuario mediante ataques a la base de datos.

Solución escogida

Como solución final, la mejor forma de almacenar la información es mediante la combinación de ambas soluciones.

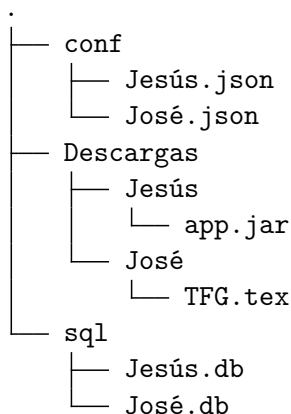


Figura 3.25: Sistema de almacenamiento. Árbol de ficheros, solución final.

Así, en el caso de los archivos compartidos, un usuario no tendrá que acceder a la base de datos cada vez que quiera recuperar un archivo recibido por un contacto, sino que directorio podrá acceder a él a través del directorio donde se encuentra almacenado. Además, almacenar los archivos recibidos fuera de la base de datos, supone una disminución en el tamaño del archivo de la base de datos, haciendo más ligera la manipulación de los datos en la base de datos.

También, como cada usuario tiene su propio fichero de base de datos, no es lógico la creación de una sola tabla donde se guardará la configuración del usuario, debido a que solamente insertaría una única fila con toda la información del usuario.

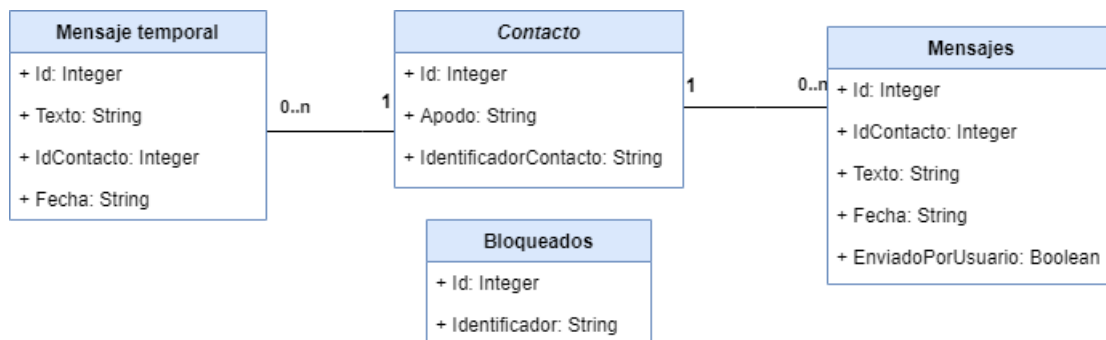


Figura 3.26: Sistema de almacenamiento. Base de datos, solución final.

Finalmente, si que es conveniente almacenar en la base de datos: los mensajes, tanto enviados como recibidos, los contactos y el listado de usuario bloqueados; por la sencillez a la hora de implementar los métodos de búsqueda, extracción, inserción y eliminación de información en la base de datos y por la rapidez a la hora de ejecutar dichas operaciones.

Seguridad

Una vez se tiene el diseño para guardar los datos generados por el usuario, es necesario protegerlos para que evitar que terceros puedan obtenerlos o incluso modificarlos. Es por ellos que se proponen las siguientes soluciones para securizarlos.

Permisos de los ficheros

Como la aplicación esta diseñada con la mentalidad de que varios usuarios, pertenecientes a un mismo sistema, pueden utilizarla, hay que modificar los permisos tanto de las carpetas como de los ficheros generados por la aplicación, para evitar que solo los usuarios puedan acceder a los ficheros que les pertenecen.

Para ello, cada usuario tendrá su propia carpeta personal, cuyo contenido solo podrá acceder el usuario legítimo de esa carpeta. Como resultado se obtiene el siguiente árbol de permisos.

```

. [drwxrwxrwx]
├── [-rwxrwxr-x] app.jar
├── [drwx-----] Jesús
│   ├── [-rw-----] conf.json
│   ├── [-rw-----] DB.db
│   └── [drwx-----] Descargas
  
```

Figura 3.27: Sistema de almacenamiento. Árbol de ficheros con permisos.

Cifrado de los datos

Paralelamente al uso de permisos con la intención de evitar a usuarios malintencionados roben, alteren o eliminen información generada por los usuarios de la aplicación, es conveniente cifrar la información generada por el usuario. Esto se considera una buena praxis, porque cualquier usuario malintencionado con suficientes privilegios podría

acceder a los archivos de los usuarios de la aplicación, obteniendo todos los mensajes escritos y recibidos por y para el usuario. Por lo tanto, si se cifran los datos de tal forma, que su recuperación sea muy costosa, evitaría que ciberdelincuentes se molestasen en robar información almacenada, pues no les saldría rentable el tiempo y dinero invertidos para la obtención de unas cuantas líneas de texto.

Finalmente, el algoritmo de cifra que se plantea utilizar en la implementación de la aplicación es **AES-256**, utilizando como clave la contraseña de autenticación del usuario. Este método es utilizado en diversas aplicaciones de almacenamiento de claves como Dashlane, Lastpass y Bitwarden porque de esta forma la seguridad del cifrado reside en la contraseña utilizada por el usuario. Además, otra ventaja que proporciona el empleo de la contraseña del usuario como clave de cifrado, es la autenticación del usuario, pues al ser el usuario el único conocedor de la clave de cifrado, permite que solamente él o ella será capaz de descifrar todos los criptogramas y que además tengan sentido. En otras palabras, si un usuario intentase descifrar un criptograma con una clave que no corresponde a su cifrado y que este contiene la palabra «Hola». El resultado del descifrado sería una serie de caracteres como los siguientes «Y.\$2».

Solución final

La solución a implementar para el almacenamiento de los datos generados por el usuario, será la creación de una carpeta, a la cual solo tendrá acceso el usuario del sistema que creó la cuenta, donde se almacenarán: un archivo .json con los parámetros de configuración de la aplicación elegidos por el usuario, una carpeta donde se guardarán los archivos compartidos por otros usuarios y un fichero de base de datos para almacenar tanto los contactos que tiene agregado el usuario como los mensajes enviados o recibidos de cada uno de los contactos.

3.8.2. Servidor

El servidor de identificadores al igual que un usuario de la aplicación tiene su propio fichero de configuración donde almacenan los parámetros de: puerto a la escucha, el uso de UPnP y el empleo de la lista blanca o negra. Para almacenar toda esta información, se realizará de la misma forma que el cliente, guardándolo todo en un fichero «.json».

Seguridad

De la misma forma que los clientes, el fichero de configuración, la lista blanca y la lista negra, tienen que estar asegurados para evitar que puedan ser modificados. La medida de seguridad que se plantea es la modificación de los permisos de los ficheros y carpetas. Se plantea de esta forma, porque se supone que el servidor solamente será empleado por el administrador del sistema. Además, el servidor no almacenará información que se considere privilegiada, por lo que no se cree oportuno aumentar, aun más, la fortificación de los ficheros del servidor.

Permisos de los ficheros

Como se ha mencionado anteriormente los ficheros almacenan en la carpeta donde se encuentra el servidor, con la intención de que solo puedan ser modificados o en el caso del servidor ejecutados por el administrador del sistema. Es por ello, que los permisos deben de ser los siguientes:

```
. [drwx-----]
├── [-rw-----] configuracion.json
├── [-rw-----] whitelist.txt
├── [-rw-----] blacklist.txt
└── [-rwx-----] servidor.jar
```

Figura 3.28: Permisos de los ficheros del servidor de identificadores.

3.9 Autenticación de los usuarios

Debido a que la autenticación de los usuarios es un punto indispensable en cualquier tipo de aplicación. Se desea implantar un sencillo sistema de inicio de sesión, basado en el empleo de las credenciales de usuario y contraseña.

La imagen muestra una interfaz de usuario para iniciar sesión. El fondo es negro. En la parte superior, el título "Iniciar sesión" está escrito en blanco. Debajo del título, hay dos campos de entrada de texto blancos. El primer campo está etiquetado "Usuario:" y el segundo "Contraseña:". Debajo de los campos, hay un botón naranja con el texto "Iniciar sesión" en blanco.

Figura 3.29: Pantalla de autenticación de usuarios.

Para poder autenticar un usuario, la aplicación debe de almacenar previamente las credenciales en un lugar determinado en el equipo. Por un lado, el nombre del usuario será almacenado como nombre de la carpeta de la cuenta registrada, ya que solamente el usuario del sistema que ha creado dicha cuenta será capaz de modificar la carpeta donde se guardaran todos sus datos. Por otro lado, la contraseña no será guardada directamente, sino que se archivará el *hash* resultante de emplear una función resumen, introduciendo como parámetro la combinación del nombre de usuario, la contraseña, la sal¹⁶ y la pimienta¹⁷. Este *hash* servirá para comprobar si las credenciales del usuarios son las correctas. Además, se almacenará la sal con el *hash* resultante.

Finalmente, cuando un usuario inicie sesión, el sistema comprobará si existe una carpeta cuyo nombre sea igual al que ha introducido el usuario, y también comprobará el hash calculado por la aplicación es igual al almacenado y en función a los resultados obtenidos permitirá acceder o no a las demás pantallas de la aplicación.

¹⁶Valor aleatorio que es añadido a un argumento antes de ser pasado a una función criptográfica. La sal puede ser almacenada junto con el *hash* resultante.

¹⁷La pimienta es lo mismo que la sal, pero el valor se almacena en un lugar seguro y separado del *hash*, o directamente no se almacena.

3.9.1. Seguridad

Además del sistema de almacenamiento de contraseñas, es necesario evitar que otros usuarios puedan realizar ataques de fuerza bruta sobre la aplicación para obtener la contraseña de un usuario en específico. Es por ello, que se penalizará con un tiempo de espera a aquellos usuarios que han escrito varias veces las credenciales incorrectamente, evitando que usuarios mal intencionados puedan acceder a la cuenta de un usuario mediante prueba y error. Además, la aplicación comprobará los permisos del usuario que esta ejecutando la aplicación y la cuenta a la que se pretende acceder. De esta forma, si un usuario del sistema consigue la contraseña de una cuenta de la aplicación, pero no tiene los permisos, a nivel de fichero, para acceder a ella, la aplicación denegará al usuario su acceso.

3.10 Interfaz

A fin de poder interactuar con nuestra aplicación, hemos de diseñar una interfaz gráfica. Esta interfaz se diseñará tanto para el usuario final como para el administrador del servidor. La interfaz deberá cumplir con los requisitos mínimos de usabilidad, siendo intuitiva y familiar al consumidor del servicio.

3.10.1. Aplicación

Para empezar con el desarrollo de la interfaz, es necesario identificar el usuario final de la aplicación. En este caso, el usuario final de la aplicación sería aquel que tiene una cierta soltura con el uso de la aplicaciones de mensajería instantánea y un evidente conocimiento informático. Es por ello, que el diseño de la interfaz se basa en la estructura de las aplicaciones Whatsapp y Telegram en su versión de escritorio.

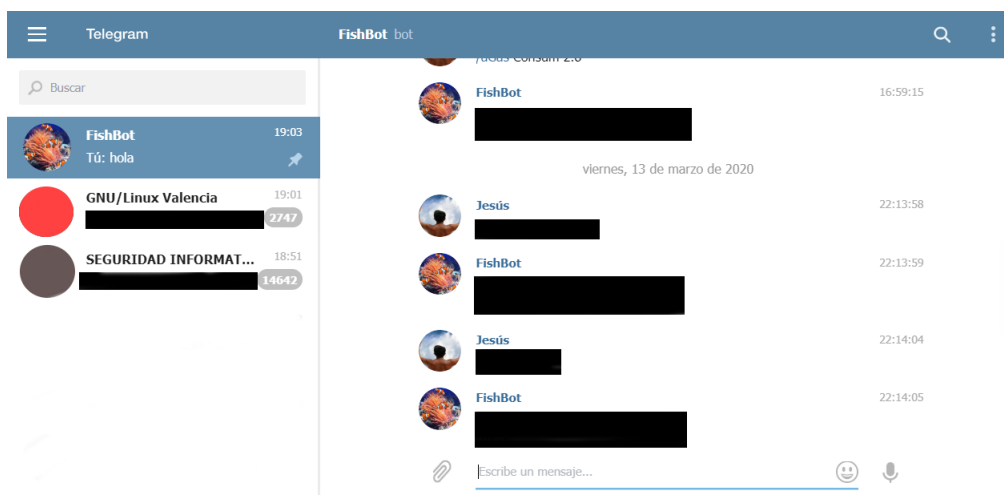


Figura 3.30: Interfaz de Telegrama.

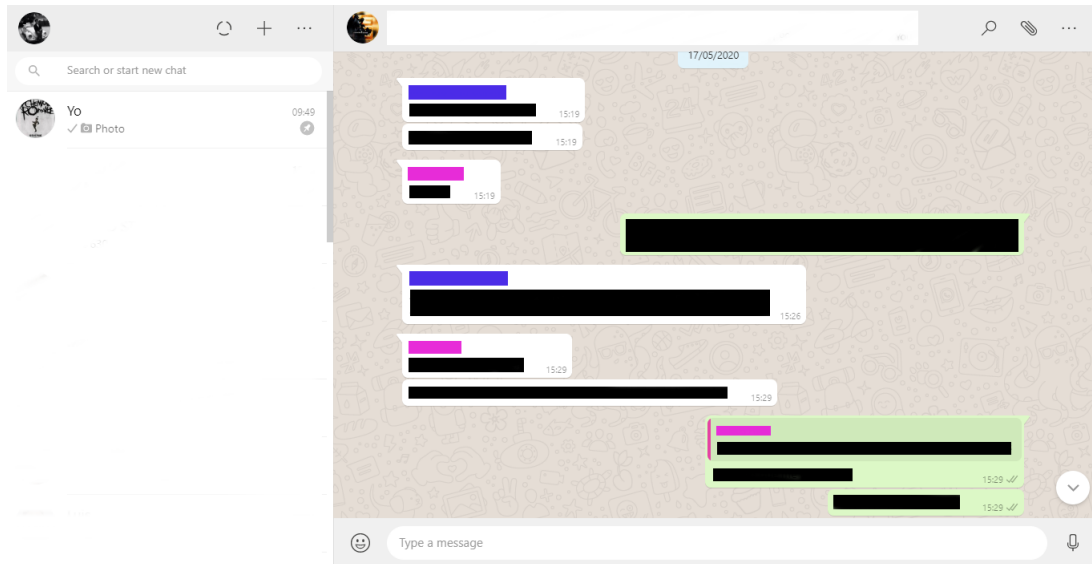


Figura 3.31: Interfaz de Whatsapp.

A partir de estas imágenes, y extrayendo los elementos comunes a ambas, se extrae el siguiente boceto de las pantallas de la aplicación:

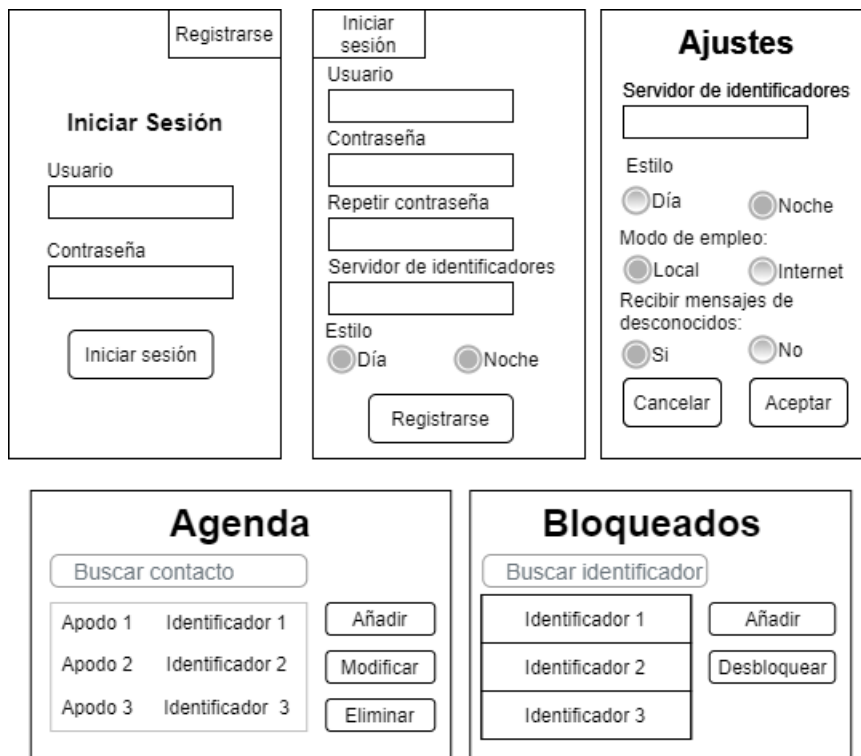


Figura 3.32: Pantallas aplicación.

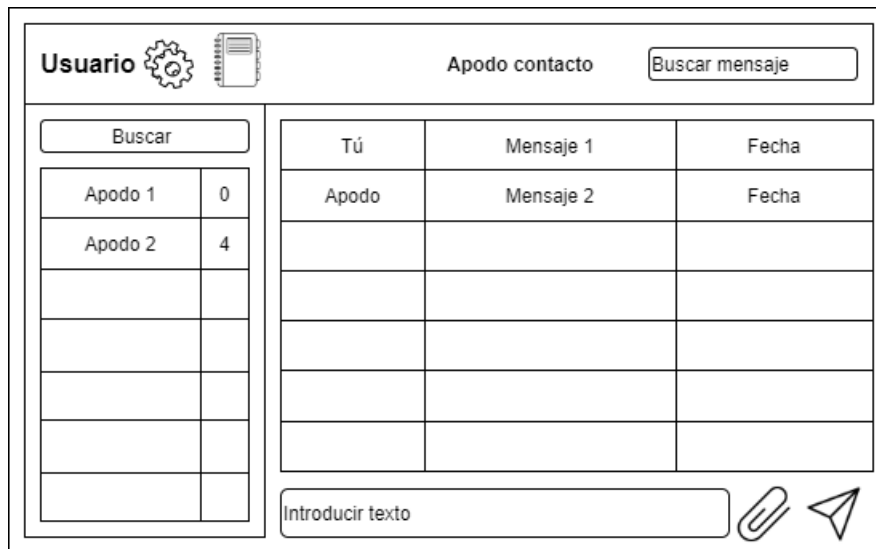


Figura 3.33: Pantalla chat aplicación.

Como se puede observar en la segunda imagen, se conserva la columna izquierda que emplea Whatsapp y Telegram destinada a mostrar los chats abiertos. Además, se mantiene la estructura de Telegram web para mostrar los mensajes intercambiados entre usuarios y un panel horizontal en el que se muestra tanto la información del contacto como la del usuario de la aplicación.

3.10.2. Servidor

En el caso del servidor de identificadores, no se considera necesaria una interfaz gráfica debido a que el administrador del servidor tendrá unos conocimientos informáticos más avanzados, seguido de un buen dominio de la terminal de comandos. Por tanto, la interfaz resultante, será una sencilla interfaz de comandos, donde el administrador será capaz de realizar las pocas opciones que proporciona el servidor.

```

Menú
=====
0) Cerrar programa.
1) Añadir identificador a la WhiteList.
2) Eliminar identificador de la WhiteList.
3) Añadir identificador a la BlackList.
4) Eliminar identificador de la BlackList.
=====
Opción:

```

Figura 3.34: Interfaz del servidor de identificadores.

CAPÍTULO 4

Implementación

En este capítulo se mostrarán las implementaciones de ciertos aspectos críticos o esenciales del sistema con el fin de entender mejor como funcionan ciertos aspectos de la aplicación.

4.1 Diffie-Hellman y AES-256 (CBC)

En el apartado 3.4.5 Confidencialidad se habló de la generación de claves efímeras con la finalidad de conseguir *Perfect Forward Secrecy*. En este apartado se detallará las acciones que tienen que realizar cada una de las partes que componen la comunicación, las cuales se llaman «Alice» y «Bob», con el fin de obtener una clave efímera.

1. Alice genera un par de claves mediante curvas elípticas, después a través del método ECDH produce una clave pública, que procede a enviar a a Bob y se espera a recibir la clave pública de Bob.

```
//Generación del par de claves
KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("EC");
aliceKpairGen.initialize(tamanoBuffer);
KeyPair aliceKpair = aliceKpairGen.generateKeyPair();
//Generación clave pública
KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("ECDH");
aliceKeyAgree.init(aliceKpair.getPrivate());
alicePubKeyEnc alicePubKeyEnc = aliceKpair.getPublic().getEncoded();
//Transmisión de la clave
enviaByteArray(socket, alicePubKeyEnc);
bobPubKeyEnc = recibeByteArray(socket);
```

Figura 4.1: Generación de la clave pública ECDH por parte de Alice.

2. Bob a partir de la clave pública de Alice obtiene todos los parámetros necesarios para generar su clave pública, una vez generada procede a enviársela a Alice.

```

//Obtención de datos de la clave pública de Alice
bobKeyFac = KeyFactory.getInstance("EC");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(ApubKeyEnc);
PublicKey alicePubKey = bobKeyFac.generatePublic(x509KeySpec);
mr.addLogLine("ECDH: [B] Sacando datos de la clave de A",false);
ECParameterSpec ecParamFromAlicePubKey = ((ECPublicKey)alicePubKey).getParams();
mr.addLogLine("ECDH: [B] Generando par de claves",false);
//Generando un par de claves
KeyPairGenerator bobKpairGen = KeyPairGenerator.getInstance("EC");
bobKpairGen.initialize(ecParamFromAlicePubKey);
KeyPair bobKpair = bobKpairGen.generateKeyPair();
//Generación de la clave pública para ECDH
KeyAgreement bobKeyAgree = KeyAgreement.getInstance("ECDH");
bobKeyAgree.init(bobKpair.getPrivate());
bobPubKeyEnc = bobKpair.getPublic().getEncoded();
//Transmisión de la clave
enviaByteArray(socket,bobPubKeyEnc);

```

Figura 4.2: Generación de la clave pública ECDH por parte de Bob.

- Una vez que Alice y Bob han obtenido las claves públicas de su opuesto, proceden a generar el secreto compartido.

```

//Alice
KeyFactory aliceKeyFac = KeyFactory.getInstance("EC");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(bobPubKeyEnc);
PublicKey bobPubKey = aliceKeyFac.generatePublic(x509KeySpec);
aliceKeyAgree.doPhase(bobPubKey, true);
//Bob
bobKeyAgree.doPhase(alicePubKey, true);

```

Figura 4.3: Generación del secreto compartido.

- A partir del secreto compartido generan el cifrador y descifrador del algoritmo AES en modo CBC.

```

//Generación de la clave
SecretKeySpec AesKey = new SecretKeySpec(bobSharedSecret, 0, 16, "AES");
//Obtebniendo los parametros para AES
AlgorithmParameters aesParams = AlgorithmParameters.getInstance("AES");
byte[] encodedParams = recibeByteArray(socket,mr);
aesParams.init(encodedParams);
//Generación del cifrador
cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, AesKey, aesParams);
//Generación del descifrador
decipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
decipher.init(Cipher.DECRYPT_MODE, AesKey, aesParams);

```

Figura 4.4: Generación de cifrador y descifrador.

Finalmente, si el lector está interesado, en el apéndice *A Código de la implementación* se muestran tanto la implementación de los métodos de envío y recepción de un conjunto de *bytes* como de los métodos para el envío y recepción de mensajes.

4.2 Envío y recepción de archivos

Para poder transmitir archivos de gran tamaño entre usuarios se han implementado los siguientes métodos.

```
byte[] buffer = new byte[tamanyoBuffer];
File archivo = new File(rutaArchivo);
fis = new FileInputStream(archivo);
DIS = new DataInputStream(new BufferedInputStream(fis));

String criptograma="";
long longitudArchivo = archivo.length();
enviaMensajeAES(socket, longitudArchivo+"", cifrador);
while((contador = DIS.read(buffer))>0)
{
    criptograma = Base64.getEncoder().encodeToString(buffer);
    enviaMensajeAES(socket, criptograma, cifrador);
    longitudArchivo-=contador;
}
enviaMensajeAES(socket, "FIN", cifrador);
```

Figura 4.5: Líneas de código para la transmisión de archivos.

```
OutputStream output = new FileOutputStream(rutaDeGuardado);
long longitudArchivo = Long.parseLong(recibeMensajeAES(socket, descifrador));
byte[] bufferDescifrado = null;
String criptograma=null;

while(!(criptograma = recibeMensajeAES(socket, descifrador)).equals("FIN")){
    bufferDescifrado = Base64.getDecoder().decode(criptograma);
    if(longitudArchivo<tamanyoBuffer && longitudArchivo!=0){
        output.write(bufferDescifrado,0,(int)longitudArchivo);
    }
    else{output.write(bufferDescifrado); }
    longitudArchivo-=bufferDescifrado.length;
}
```

Figura 4.6: Líneas de código para la recepción de archivos.

Como se puede ver en las anteriores imágenes, primero se envía al destinatario la longitud del archivo, almacenada en una variable de tipo *Long*, posteriormente se procede a enviar el archivo en bloques de un tamaño predefinido (2048 *bytes*). Estos bloques son codificados en base 64¹ para evitar errores de relleno. Sin embargo, esto aumenta el

¹Base64 es una técnica de codificación y decodificación utilizada para convertir datos binarios a un formato de texto del Estándar Americano para el Intercambio de Información (ASCII), y viceversa.

tamaño del bloque a enviar resultando en un aumento del tiempo de envío del fichero. Finalmente, para evitar posibles fallos, se le comunica al usuario que la transmisión del archivo ha sido completada.

Además, merece la pena subrayar que la limitación del tamaño del archivo a enviar es el número máximo que se puede almacenar en una variable de tipo *Long*, esto es 9223372036854775807 *bytes*, equivalente a 7 *hexabytes*.

4.3 Generación del identificador de los usuarios

En la subsección 3.6.2 *Servidor para la resolución de identificadores* se explicó la importancia de la creación de un identificador único para cada usuario registrado en la aplicación para así poder evitar colisiones en el servidor de identificadores.

```
String userID = stringSha256(usuario+randomString()+fecha());
```

Figura 4.7: Generación del identificador de un usuario.

Como se puede ver en la figura anterior, el identificador del usuario es el resultado de la combinación del nombre del usuario, más una cadena de caracteres de veinte valores aleatorios (A.6 *Método de generación de una cadena de caracteres de bytes aleatorios.*) y más la fecha (A.9 *Método de generación de fecha.*) de la creación del usuario, introducidos como parámetros de una función resumen.

4.4 Generación y comprobación del *hash* del usuario

Un punto crítico en la identificación de los usuarios es la comprobación de la contraseña, que con el fin de evitar almacenarla, se guarda el resultado de una función resumen. Esta función resumen recibe como parámetro la combinación del nombre del usuario, la contraseña del usuarios, la sal² (A.7 *Método de generación de una cadena de caracteres alfanuméricos aleatorios.*) que se almacenará en el archivo «conf.json», y la pimienta³ (A.8 *Método de generación de un entero aleatorio.*) que no será almacenada.

```
String s = MetodosReusables.randomAlfaNumericaString(10);
int pepper = MetodosReusables.randomInt(0,10);
String hash = stringSha256(usuario+contrasena+s+pepper);
```

Figura 4.8: Generación del *hash* del usuario.

Finalmente, para comprobar si la contraseña escrita por el usuario es la contraseña empleada en la creación de la cuenta, se combinan: el usuario nombre de la cuenta escrito por el usuario, la contraseña escrita por el usuario y la sal almacenada en el fichero «conf.json». Finalmente, se combina la pimienta y comprueba el *hash* resultante.

²Cadena de diez caracteres alfanuméricos aleatorios

³Un número aleatorio del 0 al 10

```
public static boolean compruebaHash(String nombreUsuario,String contrasena,
String hashPasswd, String salt){

String aux = nombreUsuario+contrasena+salt;
for(int i = 0; i < 10; i++){
    if(hashPasswd.equals(mr.stringSha256(aux+i))){return true;}
}
return false;
}
```

Figura 4.9: Comprobación del *hash* del usuario.

4.5 Autenticación entre clientes

La autenticación entre usuarios es un punto muy crítico en la aplicación, debido a que un mal diseño o una mala implementación podría permitir la realización de ataques de suplantación de identidades de los usuarios. Como se mencionaba en 3.4.5 *Autenticación* para poder autenticar a los clientes se implementará el protocolo *CHAP*. Como ejemplo y para facilitar la explicación, se pone el contexto de que Bob se quiere autenticar frente a Alice.

```
public boolean autenticaContacto() throws IOException{
String valorChallenge = randomAlfaNumericaString(randomInt(10,20));
enviaMensaje(mr.creaOperacionJSON("CHALLENGE", valorChallenge).toString());
String challengeCalculado = mr.stringSha256(c.getSecreto()+valorChallenge);
String respuestaChallenge = recibeMensaje();
if(challengeCalculado.equals(respuestaChallenge)){
    enviaMensaje(mr.creaOperacionJSON("CHALLENGE_ACEPTADO", "").toString());
    return true;
}else{
    enviaMensaje(mr.creaOperacionJSON("FIN",
"El challenge calculado y recibido no coinciden.").toJSONString());
    return false;
}
}
```

Figura 4.10: Generación del reto.

```
public boolean challengeRecibido(String randomStringRecibida, JSONParser parseador)
throws IOException, ParseException{

    String respuestaChallenge = mr.stringSha256(c.getSecreto()+randomStringRecibida);
    enviaMensaje(respuestaChallenge);
    JSONObject respuestaChallengeJSON = (JSONObject) parseador.parse(recibeMensaje());
    if(!mr.compruebaHashOperacion(respuestaChallengeJSON)){
        return false;
    }
    String operacion = (String) respuestaChallengeJSON.get("operacion");
    String valor = (String) respuestaChallengeJSON.get("valor");
    if(operacion.equals("CHALLENGE_ACEPTADO")){
        return true;
    }else{
        return false;
    }
}
```

Figura 4.11: Respuesta al reto.

Como se puede ver en las imágenes anteriores, Alice se genera un valor alfanumérico de longitud aleatoria, entre diez y veinte caracteres, que es enviado a Bob. Bob, genera un *hash* a partir de la cadena de caracteres recibida y el secreto precompartido que conocen Alice y Bob. Este *hash* es enviado Alice, quién genera el suyo propio, compara los resultados e informa a Bob si la respuesta enviada coincide con la generada o no. De la misma forma, se emplean los mismos métodos para autenticar a Alice frente a Bob y así ambos saber con quién están hablando.

4.6 AES

En el caso de la base de datos, los datos del usuario deben permanecer cifrados para evitar que ciberdelincuentes sean capaz de obtener lo que hay almacenado a pesar de haber capturado su contenido. Es por ello, que se ha implementado un cifrador y descifrador simétrico AES en modo CBC.

```

1 public AES(String contraseña){
2     MessageDigest sha = null;
3     byte[] iv = mr.stringSha256(contraseña).getBytes();
4     if(iv.length!=16){
5         byte[] aux = new byte[16];
6         for(int i = 0; i<aux.length;i++){
7             aux[i] = iv[i];
8         }
9         iv = aux;
10    }
11    IvParameterSpec ivspec = new IvParameterSpec(iv);
12    try {
13        byte[] key = contraseña.getBytes("UTF-8");
14        sha = MessageDigest.getInstance("SHA-256");
15        key = sha.digest(key);
16        key = Arrays.copyOf(key, 32);
17        SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
18
19        cifrador = Cipher.getInstance("AES/CBC/PKCS5Padding");
20        cifrador.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
21
22        descifrador = Cipher.getInstance("AES/CBC/PKCS5Padding");
23        descifrador.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
24    }catch (Exception ex) {
25        mr.addLogLine("AES: (ERROR) [AES] No se han podido generar el
26        cifrador y el descifrador.\n"+ex.toString(), true);
27    }
28 }

```

Figura 4.12: Generación de cifrado y descifrador AES.

Como se puede ver en la figura anterior, a partir de la contraseña del usuario se genera un vector de inicialización tras aplicar a la contraseña una función resumen, obteniendo los 16 primeros *bytes*. Además, la clave para generar el cifrador y el descifrador, es el resultado de aplicar un resumen de la contraseña del usuario, obteniendo los primeros 256 *bits*. Paralelamente, los métodos de cifrado y descifrado son los siguientes.

```

public String cifrar(String textoPlano){
    String res = "";
    try {
        res = Base64.getEncoder().encodeToString(cifrador.doFinal(
            textoPlano.getBytes("UTF-8")));
    } catch (Exception ex) {
        mr.addLogLine("AES: (ERROR) [cifrar] No se ha cifrar el mensaje.\n"+
            ex.toString(), true);
    }
    return res;
}

```

Figura 4.13: Método para el cifrado de texto.

```
public String descifrar(String criptograma){
    String res = "";
    try {
        res = new String(descifrador.doFinal(
            Base64.getDecoder().decode(criptograma)), "UTF-8");
    } catch (Exception ex) {
        mr.addLogLine("AES: (ERROR) [cifrar] No se ha descifrar el mensaje.\n"+
            ex.toString(), true);
    }
    return res;
}
```

Figura 4.14: Método para el descifrado de texto.

Como se puede ver en las figuras anteriores, antes de cifrar cualquier fragmento de texto se aplica la codificación en base 64 para evitar errores de relleno durante el cifrado, aumento el tamaño del criptograma que se desea almacenar. Finalmente, se ha escrito en la sección A.8 *Cifrado de la base de datos* se han escrito los métodos para el almacenamiento de la información en la base de datos, donde se emplean los métodos de cifrado y descifrado.

CAPÍTULO 5

Implantación

5.1 Instalación de Java

Para poder instalar correctamente **Java SE Development Kit 8u251** es necesario descargarlo de la [página web oficial](#). Además, el proceso de instalación varía de un sistema operativo a otro.

5.1.1. Windows

Para instalar Java SE Development Kit 8u251 en Windows, el lector solo tiene que descargarse el producto para Windows 64 o 32 bits, dependiendo de la arquitectura de su ordenador, ejecutar el archivo .exe y seguir la guía de instalación mostrada por el ejecutable.

5.1.2. Ubuntu

El proceso de instalación de Java en 8 en Linux es más complejo y más diverso. De hecho, existen varios modos de instalación: automático y manual. Si a la hora de instalarlo automáticamente se produce algún fallo o el usuario no es capaz de ejecutar la aplicación, se recomienda encarecidamente que desinstale Java, en el caso de que la instalación se haya realizado correctamente, y realiza la instalación manualmente.

Instalación automática

Para poder instalar automáticamente Java, es necesario bajarse el producto .rpm para su posterior conversión en un archivo .deb y así poder instalarlo, como se indica en las siguientes instrucciones:

1. Descargar el producto «Linux x64 RPM Package» 5.1 Instalación de Java.

2. Instalamos Alien

```
$ sudo apt-get install alien -y
```

3. Transformamos «Alien» a través del siguiente comando.

```
$ sudo alien --script jdk-8u251-linux-x64.rpm
```

4. Instalamos el paquete.

```
$ sudo dpkg -i jdk1.8_1.8.0251-1_amd64.deb
```

Instalación manual

La instalación manual consisten en la descarga del producto .tar.gz, extraerlo en la carpeta por defecto de Java y posteriormente configurar el sistema para el uso de los archivos Java que se han extraído previamente.

1. Descargar el producto «Linux x64 Compressed Archive» 5.1 Instalación de Java.
2. Crear y acceder la carpeta por defecto de Java.

```
$ sudo mkdir /usr/lib/jvm
$ cd /usr/lib/jvm
```

3. Extraer el archivo comprimido

```
$ sudo tar -xvzf ~/Downloads/jdk-8u251-linux-x64.tar.gz
```

4. Editar, con privilegios de administrador, el fichero «/etc/environment». Añadiendo a la variable «PATH» la siguiente línea de texto.

```
:/usr/lib/jvm/jdk1.8.0_251/bin:/usr/lib/jvm/jdk1.8.0_251/db/bin:
/usr/lib/jvm/jdk1.8.0_251/jre/bin
```

y finalmente, declarar las nuevas variables de entorno añadiendo las siguientes líneas al fichero.

```
J2SDKDIR="/usr/lib/jvm/jdk1.8.0_251"
J2REDIR="/usr/lib/jvm/jdk1.8.0_251/jre"
JAVA_HOME="/usr/lib/jvm/jdk1.8.0_251"
DERBY_HOME="/usr/lib/jvm/jdk1.8.0_251/db"
```

5. Informar a nuestro sistema operativo sobre los nuevos caminos que hemos instalado.

```
$ sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/lib/jvm/jdk1.8.0_251/bin/java" 0
$ sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/lib/jvm/jdk1.8.0_251/bin/javac" 0
$ sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_251/bin/java
$ sudo update-alternatives --set javac /usr/lib/jvm/jdk1.8.0_251/bin/javac
```

6. Reiniciar el ordenador y comprobar la versión de java, obteniendo un mensaje como el siguiente.

```
user1@ubuntu:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
```

5.2 Librerías

Durante el transcurso del capítulo 3 *Análisis y diseño de la solución* se fueron proponiendo diferentes soluciones como la creación de objetos JSON para el envío de mensajes, el uso de base de datos para almacenar los datos de los usuarios y el uso de UPnP para la redirección de puertos. Para estas soluciones no existen métodos en Java SDK, por lo que es necesario descargarlas de terceros.

5.2.1. SQLite

La librería oficial empleada para crear la base de datos SQLite puede ser descargada del siguiente [enlace](#).

5.2.2. JSON

Como no existe una librería oficial para crear objetos JSON, hay que decidir cual es la más indicada para este trabajo. Para este trabajo se ha decidido que la mejor librería a utilizar es «JSON.simple», pues según el artículo «The Ultimate JSON Library: JSON.simple vs GSON vs Jackson vs JSONP» escrito por Josh Dreyfuss [36] es la que mejor analiza sintácticamente objetos JSON, tanto grandes como pequeños. JSON.Simple puede ser descargada a partir del siguiente [enlace](#).

5.2.3. WeUPnP

Al igual que JSON, no existe ninguna librería oficial para UPnP que contenga implementado el redireccionamiento de puertos. Todas las librerías que existen hoy en día varían desde las más generales y al mismo tiempo más complejas, hasta las más específicas y más fáciles de usar. Por lo tanto, se ha elegido «WeUPnP» porque es una librería específicamente diseñada para el redireccionamiento de puertos y a la vez bastante sencilla de utilizar. La librería se puede descargar a partir del siguiente [enlace](#).

Finalmente, las librerías tiene que estar almacenadas en una carpeta llamada «lib», teniendo los permisos necesarios para que ninguna librería pueda ser eliminada ni modificada. Resultando en los siguientes árboles de ficheros para el cliente y para el servidor.

```

. [drwxrwxrwx]
├── [-rwxrwxr-x] app.jar
├── [drwx-----] Jesús
│   ├── [-rw-----] conf.json
│   ├── [-rw-----] DB.db
│   └── [drwx-----] Descargas
└── [drwxrwxr-x] lib
    ├── [-r--r--r--] json-simple-1.1.jar
    ├── [-r--r--r--] sqlite-jdbc-3.30.1.jar
    └── [-r--r--r--] weupnp-0.1.4.jar

```

Figura 5.1: Árbol de fichero de la aplicación con librerías.

```

. [drwx-----]
├── [-rw-----] configuracion.json
├── [-rw-----] whitelist.txt
├── [-rw-----] blacklist.txt
├── [drwx-----] lib
│   ├── [-r-----] json-simple-1.1.jar
│   └── [-r-----] weupnp-0.1.4.jar
└── [-rwx-----] servidor.jar

```

Figura 5.2: Árbol de fichero del servidor con librerías.

CAPÍTULO 6

Validación

En este capítulo se validarán ciertos aspectos importantes de la aplicación que han sido desarrollados durante el transcurso de este trabajo.

6.1 Direccionamiento de puertos mediante UPnP

En esta sección para comprobar que el direccionamiento de puertos a través de los métodos, integrados en la aplicación, funcionan correctamente. Se ejecutará tanto una instancia de la aplicación, iniciando una sesión creada previamente con el modo «Internet», como una instancia del servidor, habilitando el modo «UPnP» a través del archivo «configuracion.json».

UPnP Portmap Table



Active	Protocol	Int. Port	Ext. Port	IP Address	Delete
✓	TCP	4949	4949	192.168.1.142	
✓	TCP	4950	4950	192.168.1.142	

Figura 6.1: Tabla de direccionamiento de puertos con UPnP.

El resultado mostrado en la imagen anterior, presenta la realización de dos direccionamientos de puertos a una dirección IP específica, pues tanto el servidor como la aplicación se encuentran bajo la misma dirección IP. Por ende, se puede aprobar el funcionamiento de UPnP integrado en la aplicación y en el servidor.

6.2 Listas blanca y negra del servidor de identificadores

Con el fin de validar el correcto funcionamiento del empleo de las listas blanca y negra, se han realizado dos pruebas diferentes, requiriendo de dos clientes, cada uno instanciado en un ordenador diferente. El cliente A con la dirección IP «192.168.1.142» y el cliente B con la dirección IP «192.168.1.143».

En la primera prueba se comprueba el funcionamiento uso de la lista blanca, haciendo que solamente una dirección IP determinada pueda obtener y aportar información al servidor de identificadores. Para ello, se habilita el uso de lista blanca en el fichero «configuracion.json» y se añade la dirección IP «192.168.1.143» al archivo whitelist.txt. Después, ejecutamos el servidor e intentamos acceder con ambos usuarios.



Figura 6.2: Mensaje de bloqueo de la aplicación.

```
22:22:26: MAIN: Conexion realizada con: 192.168.1.142
22:22:29: OperacionesClientesThread: Usuario bloqueado, no se encuentra en la lista blanca
22:22:29: OperacionesClientesThread: Fin de la conexión.
```

Figura 6.3: Servidor de identificadores. Resultado al obtener e informar al servidor.

En las dos primeras imágenes se ha mostrado como el usuario *A* no ha sido capaz de obtener información del servidor. Además, en la siguiente imagen, se muestra como el usuario *B* ha sido capaz de registrar su identificador en el servidor de identificadores.

```
22:23:04: MAIN: Conexion realizada con: 192.168.1.143
22:23:04: OperacionesClientesThread: Se ha registrado '13e11b40b26141e99107a09063b76911452dd259f2a8accac90befa554d04b57'
22:23:04: OperacionesClientesThread: Fin de la conexión.
```

Figura 6.4: Registro del servidor de mensajes de una IP no bloqueada.

En la segunda prueba se añade a la lista negra la dirección IP del usuario *A* para comprobar que no puede obtener ni informar al servidor de identificadores.

```
1:14:57: MAIN: Conexion realizada con: 192.168.1.142
1:15:00: OperacionesClientesThread: Usuario bloqueado, se encuentra en la lista negra
1:15:00: OperacionesClientesThread: Fin de la conexión.
```

Figura 6.5: Lista Blanca. Registro del servidor de mensajes de una IP bloqueada.

```
1:15:09: MAIN: Conexion realizada con: 192.168.1.143
1:15:09: OperacionesClientesThread: Se ha registrado '67bbece564b5bfe6e73f
1:15:09: OperacionesClientesThread: Fin de la conexión.
```

Figura 6.6: Lista Blanca. Registro del servidor de mensajes de una IP no bloqueada.

Como se puede comprobar en las imágenes obtenidas del servidor de identificadores, el servidor ha bloqueado al usuario que se encontraba en la IP almacenada en la lista negra, permitiendo que el usuario *B* se conectase sin problemas.

Finalmente, se puede dar por correcto, el funcionamiento del empleo de listas blanca y negra del servidor de identificadores.

6.3 Análisis del tráfico

En esta sección se desea comprobar si la información transmitida por la aplicación, que viaja a través de la red, se encuentra cifrada durante su transmisión. Con ese fin, a través de la herramienta «Wireshark», se analizará el tráfico resultante de enviar mensajes al servidor de identificadores y a otro cliente.

Con respecto a los mensajes transmitidos al servidor, en las siguiente imágenes se puede comprobar que a las 01:55:03 se han enviado 316 bytes de información cifrada al servidor de identificadores. Este mensaje se encontraba cifrado durante su envío y el servidor ha sido capaz de descifrarlo e interpretar su información.

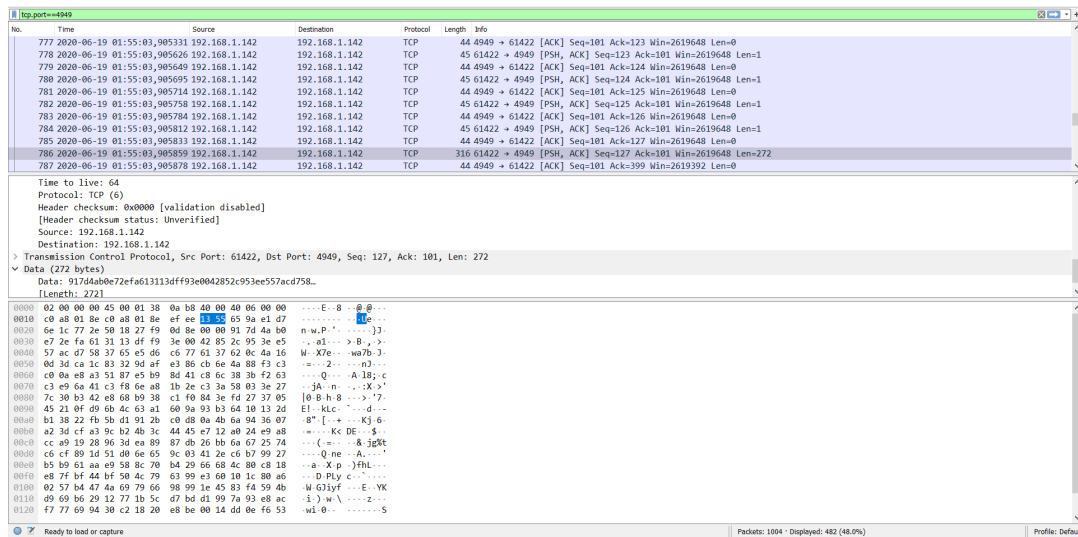


Figura 6.7: Análisis de los paquetes transmitidos al servidor de identificadores

```

1:54:10: MAIN: Conexion realizada con: 192.168.1.143
1:54:10: OperacionesClientesThread: Se ha registrado '67bbece564b5bfe6'
1:54:10: OperacionesClientesThread: Fin de la conexión.

1:55:03: MAIN: Conexion realizada con: 192.168.1.142
1:55:03: OperacionesClientesThread: Se ha registrado 'd2f88f65250fc788'
1:55:03: OperacionesClientesThread: Fin de la conexión.

1:55:10: MAIN: Conexion realizada con: 192.168.1.143
1:55:10: OperacionesClientesThread: Se ha registrado '67bbece564b5bfe6'
1:55:10: OperacionesClientesThread: Fin de la conexión.

```

Figura 6.8: Registro del servidor de mensajes.

Con respecto a la comunicación entre clientes, se puede ver en las siguientes imágenes, el mensaje enviado a las 2:27:01 al contacto Z se encontraba cifrado durante su transmisión.



Figura 6.9: Captura de pantalla del mensaje enviado al contacto Z.

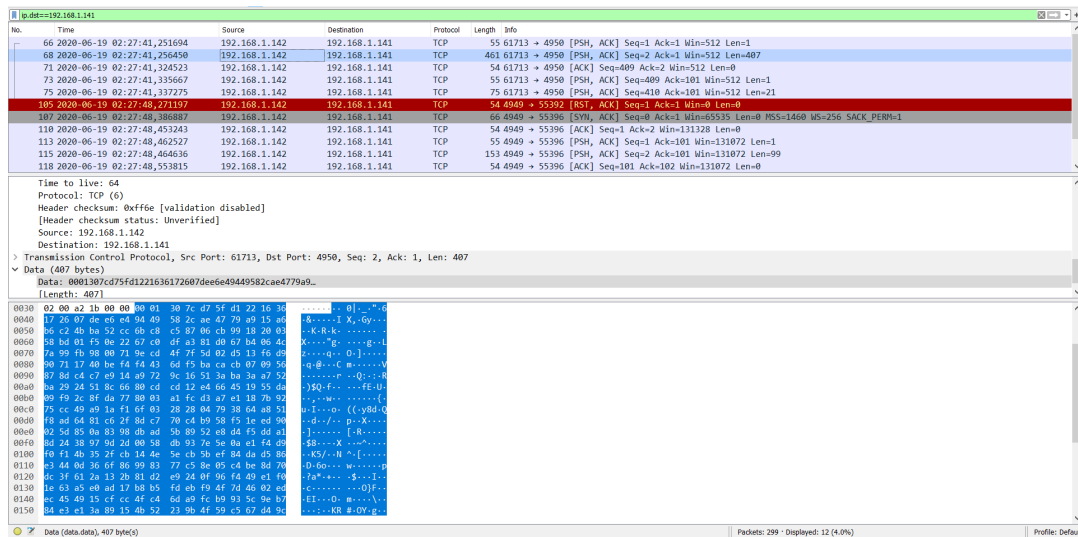


Figura 6.10: Análisis de los paquetes transmitidos al cliente Z.

Finalmente, se puede dar por valido el cifrado de la información transmitida por la aplicación.

6.4 Cifrado de la base de datos

En esta sección se validará el cifrado de los datos almacenados en el fichero de base de datos, mediante la herramienta «sqlite3». Para comprobarlo, se mostrarán los datos descifrados que el usuario puede encontrar durante la ejecución de la aplicación, y posteriormente se mostrarán como se encuentran los datos almacenados en el fichero de la base de datos.

Primero, se muestra, en las siguiente imágenes, la información en texto plano genera por el usuario a través de la aplicación.



Figura 6.11: Mensajes enviados al contacto *b*.

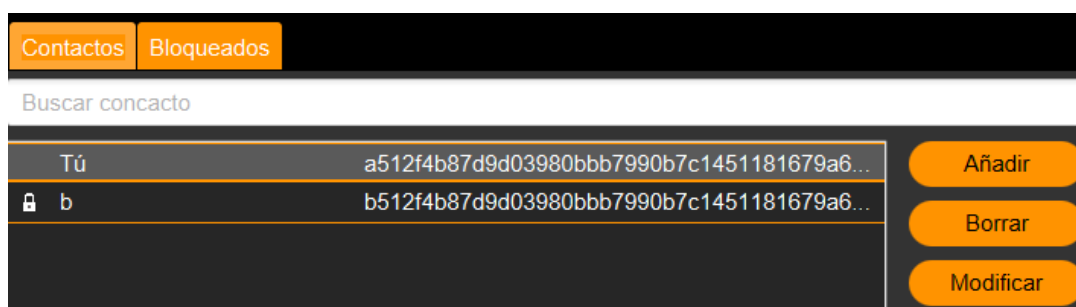


Figura 6.12: Contactos almacenados en la aplicación.

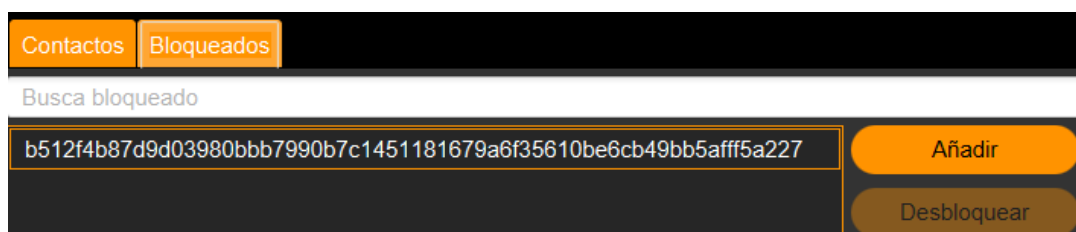


Figura 6.13: Usuarios bloqueados y almacenados en la aplicación.

Después, mediante las siguientes imágenes, se muestra la información cifrada que se encuentra almacenada en la base de datos a través de la aplicación «sqlite3».

```
sqlite> select * from mensajes;
1|2|7J8LpVLMd2xjhenp51cJSp0fPw4GAaKWbnJf6A9GG2g=|5YVN1Cwm43soAUTNEBcgU3Puh0xBVINx8ShEGW3yB08=|4GHWDFFTzTZ
bm3vK01qSsg==
sqlite>
```

Figura 6.14: Mensajes cifrados y almacenados en la base de datos.

```
sqlite> select * from contactos;
1|j9/ad+AFiS/44/3HhkRwPw==|32s5WzT9+BGiaj9xDRKWyAGGjvqtId6c0dEVFsbnp0WBoY09tXpX+n027LHIseh1USyG8kQI9wD/zh
uDEtekuBr3bx122oxnk1/H/JTM+dQ=|9bPGz0QjF06JAbaoIHHe2A==|9bPGz0QjF06JAbaoIHHe2A==|83Gagc7ACYvB8S4+D117CA==
|CAGJSDzD0th5N1IFXVksYRXvUGhFBnKGOs42bIcdE=
2|Z8/Jcig1rNRb/RLJvfvVZg==|513IqQw9cRwtsmLwL4Uepn6QuToMPQxpu/Jt01Gh6nU4N7Cp0rL65g83rabC5e9IXhHTPFKh009rVs
TvSfU03p9IUxfWw+k6CasaseDwqA=|4hUXtQHdtJlbomb1GITk1g==|9bPGz0QjF06JAbaoIHHe2A==|4GHWDFFTzTZbm3vK01qSsg==
|VjB+6eE2iwmjAnlnPdvSDA==
sqlite>
```

Figura 6.15: Contactos cifrados y almacenados en la base de datos.

```
sqlite> select * from bloqueados;
1|513IqQw9cRwtsmLwL4Uepn6QuToMPQxpu/Jt01Gh6nU4N7Cp0rL65g83rabC5e9IXhHTPFKh009rVsTvSfU03p9IUxfWw+k6CasaseD
wqA=
```

Figura 6.16: Usuarios cifrados bloqueados y almacenados en la base de datos.

```
sqlite> select * from mensajesTemp;
1|s/i1K8bjYsm4gzGNz51L0COWtq7H10105Ci1Cc/XFNcuvUqAzI/cGijyZrSzog694E32ZTbBSwK+izZfIMpqQUkg89+31L09UQ+gK7
0L30YuUZ5rnsSVnqEj0PCKT/Lwi1GcSF3ZPov5hNKuIo2r3ViEetXht1T3wFN0w14asJQCzswpk8107o3hdYQWlo4p+PfbzScgDUaJmiI
k/XsSOBQXdb1DvByT8/Tfdh4yndPYZ9GUFBP38zEY9V/mVvGoog8usVf9L38t/V5xtAXA==|2
sqlite>
```

Figura 6.17: Mensajes temporales cifrados y almacenados en la base de datos.

Los valores que se muestran en las anteriores imágenes, es el resultado del cifrado y posterior conversión a base 64 de los datos generados por el usuario. Además, estos valores almacenados en la base de datos no muestran ninguna relación aparente frente a los datos en texto plano mostrados durante la ejecución de la aplicación.

Para poder demostrar que estos valores son cifrados por AES en modo CBC, se va a utilizar la web [CyberChef](#) para descifrar el mensaje enviado al *b* que se encuentra cifrado en la base de datos. El mensaje a descifrar es el siguiente:

«ec9f0ba552e60f6c6385e9e9e757094a9d1f3f0e0601a2966e725fe80f461b68».

Por un lado, la herramienta «AES Decrypt» que se encuentra en la web [CyberChef](#), requiere que el texto a descifrar se encuentre en el sistema hexadecimal. Por consiguiente se ha empleado la web [base64.guru](#) para convertir el mensaje de base64 a hexadecimal.

Base64* [copy](#) [clear](#) [download](#)

738LpVlMd2xjhenp51c7Sp0fPw46AaKlbn7f6A9GG2g=

Letters Case
Lowercase (a1b2c3) ▼

Length

For example, specify "128" to get only the first 128 characters of the hex string. Use negative numbers (eg, "-128") to get the last 128 characters

Delimiter

For example, specify a space to get "a1 b2 c3" or specify a comma to get "a1,b2,c3" (by default there is no delimiter, so it returns "a1b2c3")

Convert Base64 to Hex

Hex [copy](#) [clear](#) [download](#)

ec9f0ba552e60f6c6385e9e9e757094a9d1f3f0e0601a2966e725fe80f461b68

Figura 6.18: Conversión del mensaje de base64 a hexadecimal.

Por otro lado, la herramienta requiere tanto la clave de cifrado como el vector de inicialización en hexadecimal. Para obtener dichos valores se ha creado una función capaz de convertir un conjunto de *bytes* en una cadena de caracteres, formada por dicho conjunto de bytes en sistema hexadecimal. Además, se han añadido las siguientes líneas de código entre las líneas 16 y 17 de la figura 4.12 *Generación de cifrado y descifrador AES*.

```
System.out.println(hex(iv));  
System.out.println(hex(key));
```

Figura 6.19: Código para mostrar la clave y el IV en formato hexadecimal.

Obteniendo, tanto la clave como el IV, al iniciar la sesión como el usuario *a*.

```
30 33 61 63 36 37 34 32 31 36 66 33 65 31 35 63  
03 ac 67 42 16 f3 e1 5c 76 1e e1 a5 e2 55 f0 67 95 36 23 c8 b3 88 b4 45 9e 13 f9 78 d7 c8 46 f4
```

Figura 6.20: Clave e IV en sistema hexadecimal.

Finalmente, insertando toda la información obtenida anteriormente y ajustando la herramienta para que emplee el CBC, se obtiene el mensaje descifrado.

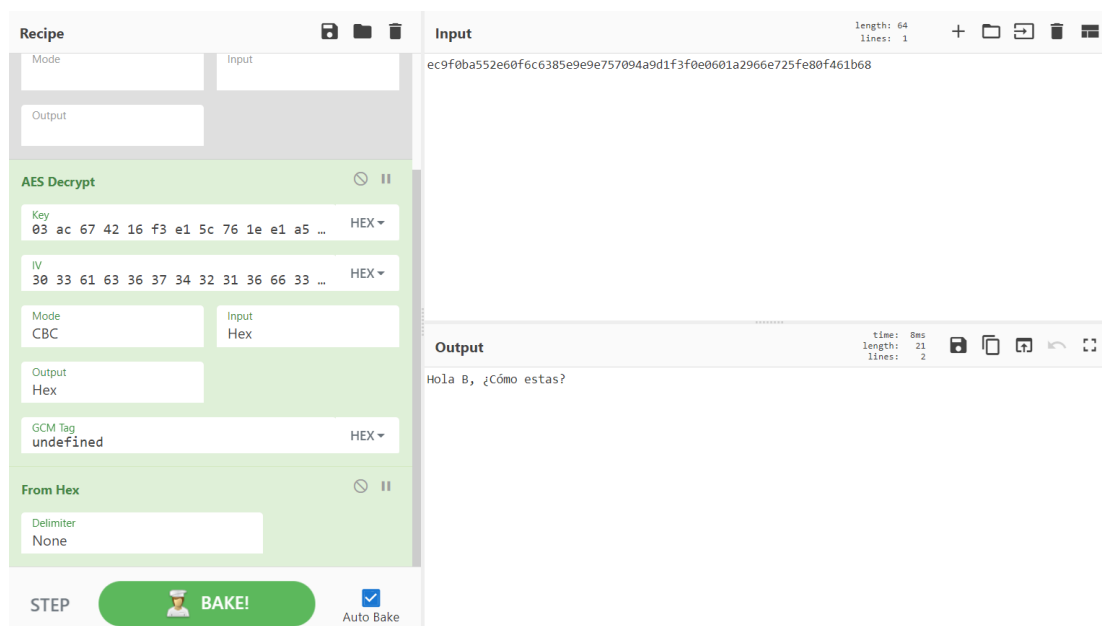


Figura 6.21: Texto descifrado a través de la web CyberChef.

Por ende, se puede dar por válido que toda la información generada por el usuario y almacenada en el fichero de la base de datos se encuentra cifrada.

CAPÍTULO 7

Conclusiones

En este capítulo se analiza el cumplimiento de los objetivos, la relación de los estudios con el trabajo realizado y la propuesta de mejoras a realizar.

7.1 Cumplimiento de objetivos

En lo que respecta al cumplimiento de los objetivos que se definieron al inicio del proyecto, cabe destacar que el principal objetivo (desarrollar una aplicación de mensajería punto a punto gratuita) se ha cumplido. Tal y como se esperaba de la aplicación punto a punto, se ha podido comprobar que los mensajes son transmitidos entre clientes, sin tener que transmitirlos a un intermediario.

Además, se planteaba evitar, en la medida de lo posible, emplear un servidor centralizado. Tal y como se ha observado y explicado, para la transmisión de mensajes es necesario al menos un servidor de identificadores. No obstante, un usuario no tiene porque estar ligado a utilizar un único servidor, pues pueden existir diferentes instancias en la red, donde cada usuario puede emplear el que más se adapte a sus condiciones o exigencias.

Por último, se estableció como objetivo poner en práctica la implementación de diferentes conceptos relacionados con la criptografía y la seguridad. Se ha demostrado que la aplicación es capaz de cifrar no sólo los datos que viajan a través de Internet, sino también toda la información del usuario, que se encuentra almacenada en local.

En conclusión, los objetivos fijados al inicio de este proyecto han sido alcanzados en su totalidad de forma satisfactoria.

7.2 Relación del trabajo desarrollado con los estudios cursados

Durante la realización de este trabajo se han puesto en práctica competencias adquiridas a lo largo de todo el grado. De entre todas las asignaturas cursadas en el grado, las que más he aplicado en el trabajo serían.

- Introducción a la informática y a la programación.
- Criptografía.
- Interfaces Persona Computador.
- Desarrollo Centrado en el Usuario.

Finalmente, es de gran importancia resaltar que a raíz al desarrollo del trabajo de fin de grado, he sido capaz de aumentar mi conocimiento en diferentes áreas de la informática que durante el transcurso del grado no se han podido mostrar en profundidad.

7.3 Trabajos futuros

Con el fin de mejorar la aplicación se propone para trabajos futuros, implementar la función de «crear grupos de contactos», pues no pudo ser implementada por falta de tiempo. Además, se propone desarrollar la aplicación para dispositivos móviles.

Bibliografía

- [1] CBR Staff Writer. 12 de junio de 2019. *Telegram Briefly Taken Offline in “Powerful” DDoS Attack*. Computer Business Review
- [2] Napier Lopez. octubre de 2019. *Facebook leak contained phone numbers for 419 million users*. thenextweb. <https://thenextweb.com/facebook/2019/09/04/facebook-leak-contained-phone-numbers-for-419-million-users/>
- [3] Paul Bischoff. 9 de octubre de 2019. *Report: 267 million Facebook users IDs and phone numbers exposed online (UPDATE: now 309 million)*. comparitech. <https://www.comparitech.com/blog/information-security/267-million-phone-numbers-exposed-online/>
- [4] Cesar Otero. 14 de marzo de 2019. *No, WhatsApp no se cayó por un ataque DDoS: Esta fue la razón*. as. https://as.com/meristation/2019/03/14/betech/1552600219_433099.html
- [5] Radical App International. *Mercury Protocol*. 2017
- [6] derechodelared. 30 de julio de 2018. *Dust, ¿un lugar seguro para mandar mensajes?*. derechodelared. <https://derechodelared.com/dust/>
- [7] 16 de enero de 2019. *Dust Privacy Statement*. usedust.com. <https://usedust.com/privacy-policy/>
- [8] Eduardo Medina, 20 de septiembre de 2018. *JavaFX 11 ya funciona oficialmente como un módulo independiente de JDK muy linux*. <https://www.muylinux.com/2018/09/20/javafx-11-modulo-independiente-jdk/>
- [9] University of Southern California. 1981. *Transmission Control Protocol*. Recuperado de <https://tools.ietf.org/html/rfc793>
- [10] J. Postel. 1980. *User Datagram Protocol*. Recuperado en <https://tools.ietf.org/html/rfc768>
- [11] Douglas Crockford. 2014. *The JavaScript Object Notation (JSON)*. Data Interchange Format. (n.d.). Recuperado en <https://tools.ietf.org/html/rfc7159>
- [12] S. Hollenbeck. 2003. *Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols*. Recuperado en <https://tools.ietf.org/html/rfc3470>
- [13] Maxon Novak. *Serialization Performance comparison*. M@X on DEV <https://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmlDataContractSerializer-xmlserializer-binaryformatter-json-newtonsoft-servicestack-text/>

- [14] Alfonso Muñoz y Jorge Ramió. 2019. Capítulo IV La seguridad de la criptografía de clave pública y algoritmo RSA. *Cifrado de las comunicaciones digitales. De la cifra clásica al algoritmo RSA*. Madrid. 0xWORD
- [15] Ed Harmoush. 18 de septiembre de 2015. *Message Integrity*. practical networking. <https://www.practicalnetworking.net/series/cryptography/message-integrity/>
- [16] Ángel Samuel Sánchez, 09 diciembre 2018. *¿Qué es una dirección IP? ¿Cómo puedo saber mi IP?* raiolanetworks. <https://raiolanetworks.es/blog/que-es-una-direccion-ip/>
- [17] Bradley Mitchell, 11 noviembre 2019. *What Does Dynamic DNS Mean?* lifewire. <https://www.lifewire.com/definition-of-dynamic-dns-816294>
- [18] Aaron Leonard. 08 septiembre, 2017 *Fundamentals of 802.11 Wireless Sniffing* CISCO. <https://www.cisco.com/c/en/us/support/docs/wireless-mobility/80211/200527-Fundamentals-of-802-11-Wireless-Sniffing.html>
- [19] Serge Malenkovich, 10 abril 2013. *¿QUÉ ES UN ATAQUE MAN-IN-THE-MIDDLE?* Kaspersky daily. <https://latam.kaspersky.com/blog/que-es-un-ataque-man-in-the-middle/469/>
- [20] Nica Latto, 26 de marzo de 2020. *What is a Sniffer and How Can You Prevent Sniffing?* AVG. <https://www.avg.com/en/signal/what-is-sniffer>
- [21] Simon Blake-Wilson, mayo de 2006. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*. Recuperado el 10 de junio de 2020 de <https://tools.ietf.org/html/rfc4492>.
- [22] *Authentication protocol* (Sin fecha). En Wikipedia. Recuperado el 23 de marzo de 2020 de https://en.wikipedia.org/wiki/Authentication_protocol
- [23] Chad R Dougherty. 31 de diciembre de 2008. *MD5 vulnerable to collision attacks*. Carnegie Mellon University. <https://www.kb.cert.org/vuls/id/836068/>
- [24] John Carl Villanueva. 30 de julio de 2009. *How Many Atoms Are There in the Universe?*. universitytoday. <https://www.universetoday.com/36302/atoms-in-the-universe/>
- [25] Xiacong Fan. *Real-Time Embedded Systems: Design Principles and Engineering Practices* Newnes.
- [26] R.L. Rivest, A. Shamir, and L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. MIT. <https://people.csail.mit.edu/rivest/Rsapaper.pdf>
- [27] Whitfield Diffie, Martin E. Hellman . *New direccions in cryptography*. IEEE. <https://ee.stanford.edu/hellman/publications/24.pdf>
- [28] National Institute of Standards and Technology. 1972. *DATA ENCRYPTION STANDARD (DES)*. emitido el 25 de octubre de 1999. <https://nvlpubs.nist.gov/nistpubs/sp958-lide/250-253.pdf>
- [29] Susan Landau, marzo de 2000. *Standing the Test of Time: The Data Encryption Standard*. University of California, Davis. <https://web.cs.ucdavis.edu/rogaway/classes/754/2002/landau-des.pdf>
- [30] National Institute of Standards and Technology. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. emitido el 26 de Noviembre de 2001. <https://www.nist.gov/publications/advanced-encryption-standard-aes>

-
- [31] Lourdes Acuayte y Cutberto Hernández. 24 de septiembre de 2012. *Cifrado Feistel* <http://cifradofeistel.blogspot.com/>
- [32] Shawn Wang. *The difference in five modes in the AES encryption algorithm*. emitido el 8 de agosto de 2019. <https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/>
- [33] Nilesh A. Lal. 7 de julio de 2017. *A Review Of Encryption Algorithms-RSA And Diffie-Hellman*. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 6 <http://www.ijstr.org/final-print/july2017/A-Review-Of-Encryption-Algorithms-rsa-And-Diffie-hellman.pdf>
- [34] Anastasios Arampatzis. 13 de marzo de 2019. *Encryption should not be seen as the ultimate answer*. VENAFI. <https://www.venafi.com/blog/how-diffie-hellman-key-exchange-different-rsa>
- [35] Douglas R. Stinson. *Cryptography, theory and practice, third edition*, Punto 3.7 University of Waterloo, Ontario, Canada. 2006.
- [36] Josh Dreyfuss. 13 de agosto de 2017. *The Ultimate JSON Library: JSON.simple vs GSON vs Jackson vs JSONP*. OverOps. <https://blog.overops.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/>

APÉNDICE A

Código de la implementación

En este anexo se encuentra la implementación de algunos métodos que aparecen en la sección 6.4 y *Cifrado de la base de datos* el capítulo 4 *Implementación*.

A.1 Conversor de un conjunto de *bytes* a hexadecimal

```
public static String hex(byte[] bytes) {
    StringBuilder result = new StringBuilder();
    for (byte aByte : bytes) {
        result.append(String.format(" %02x", aByte));
    }
    return result.toString();
}
```

Figura A.1: Implementación del conversor.

A.2 Método envío de un conjunto de *bytes*

```
public static void enviaByteArray(Socket s, byte[] msg) throws IOException{
    DataOutputStream dOut = null;
    try {
        dOut = new DataOutputStream(s.getOutputStream());
        dOut.writeInt(msg.length);
        dOut.write(msg);
        dOut.flush();
    } catch (IOException ex) {
        throw new IOException();
    }
}
```

Figura A.2: Método para el envío de un conjunto de *bytes*.

A.3 Método recepción de un conjunto de *bytes*

```

public static byte[] recibeByteArray(Socket s) throws IOException{
    byte[] message = null;
    DataInputStream dIn = null;
    if(s == null || s.isClosed() ){return null;}
    try {
        dIn = new DataInputStream(s.getInputStream());
        int length = dIn.readInt();
        if(length>0) {
            message = new byte[length];
            dIn.readFully(message, 0, message.length); // Leer el mensaje
        }
    } catch (IOException ex) {
        throw new IOException();
    }
    return message;
}

```

Figura A.3: Método para la recepción de un conjunto de *bytes*.

A.4 Método envío mensaje cifrado

```

public static void enviaMensajeAES(Socket socket, String plaintext, Cipher cipher)
throws IOException {
    byte[] msg=null;
    try {
        if(plaintext==null){
            msg = cipher.doFinal("null".getBytes("UTF-8"));
        }else{
            msg = cipher.doFinal(plaintext.getBytes("UTF-8"));
        }
    } catch (IllegalBlockSizeException ex) {
        mr.addLogLine("ECDH: (EnviaMensajesAES) IllegalBlockSizeException", true);
    } catch (BadPaddingException ex) {
        mr.addLogLine("ECDH: (EnviaMensajesAES) BadPaddingException", true);
    } catch (UnsupportedEncodingException ex) {
        mr.addLogLine("ECDH: (EnviaMensajesAES) UnsupportedEncodingException", true);
    }
    enviaByteArray(socket, msg);
}

```

Figura A.4: Método para el envío de un mensaje cifrado.

A.5 Método de recepción de mensaje cifrado

```
public static String recibeMensajeAES(Socket socket,Cipher decipher)
throws IOException {
    String res="";
    byte[] ciphertext = recibeByteArray(socket);
    byte[] plaintext = null;
    try {
        if(ciphertext==null){return null;}
        plaintext = decipher.doFinal(ciphertext);
        res = new String(plaintext, "UTF-8");
    } catch (IllegalBlockSizeException ex) {
        mr.addLogLine("ECDH: (recibeMensajeAES) IllegalBlockSizeException", true);
    } catch (BadPaddingException ex) {
        mr.addLogLine("ECDH: (BadPaddingException) IllegalBlockSizeException", true);
    } catch (UnsupportedEncodingException ex) {
        mr.addLogLine("ECDH: (BadPaddingException) UnsupportedEncodingException", true);
    }
    return res;
}
```

Figura A.5: Método para la recepción de un mensaje cifrado.

A.6 Generadores de caracteres aleatorios

A.6.1. RandomString

```
public static String randomString() {
    byte[] array = new byte[20];
    new Random().nextBytes(array);
    String generatedString = new String(array, Charset.forName("UTF-8"));
    return generatedString;
}
```

Figura A.6: Método de generación de una cadena de caracteres de *bytes* aleatorios.

A.6.2. RandomAlfaNumericaString

```
public static String randomAlfaNumericaString(int n)
{
    String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        + "0123456789"
        + "abcdefghijklmnopqrstuvwxyz";
    StringBuilder sb = new StringBuilder(n);

    for (int i = 0; i < n; i++) {
        int index = (int)(AlphaNumericString.length()*Math.random());
        sb.append(AlphaNumericString.charAt(index));
    }
    return sb.toString();
}
```

Figura A.7: Método de generación de una cadena de caracteres alfanuméricos aleatorios.

A.6.3. RandomInt

```
public static int randomInt(int min, int max){
    Random r = new Random();
    return r.nextInt((max - min) + 1) + min;
}
```

Figura A.8: Método de generación de un entero aleatorio.

A.7 Generación de fecha

```
public static String fecha(){
    SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    Calendar date = Calendar.getInstance();
    return formato.format(date.getTime());
}
```

Figura A.9: Método de generación de fecha.

A.8 Cifrado de la base de datos

A.8.1. Insertar un contacto

```
public void insertaContacto(String apodo, String identificadorContacto, String secreto)
    throws SQLException{
    String query = "INSERT INTO contactos
        (apodo, identificadorContacto ,numMensajesTotales , numMensajesSinLeer,
        estaBloqueado, secreto)
        VALUES (?,?,,?,?,?)";
    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setString(1, aes.cifrar(apodo));
        pstmt.setString(2, aes.cifrar(identificadorContacto));
        pstmt.setString(3, aes.cifrar("0"));
        pstmt.setString(4, aes.cifrar("0"));
        pstmt.setString(5, aes.cifrar("false"));
        pstmt.setString(6, aes.cifrar(secreto));
        pstmt.executeUpdate();
        pstmt.close();
        mr.addLogLine("BD: Contacto insertado correctamente", false);
    } catch (SQLException e) {
        mr.addLogLine("BD: (ERROR) Error al crear un usuario", true);
        throw new SQLException();
    }
}
```

Figura A.10: Método para cifrar e insertar un contacto en la base de datos.

A.8.2. Descifra contacto

```
private List<String[]> descifraContacto(ResultSet rs){
    List<String[]> res = new ArrayList<>();
    try {
        int nColumns = rs.getMetaData().getColumnCount();
        while(rs.next()){
            String[] rows = new String[nColumns];
            for(int i = 0; i<nColumns;i++){rows[i]="";}
            rows[0] = rs.getString("id")+"";
            rows[1] = aes.descifrar(rs.getString("apodo"));
            rows[2] = aes.descifrar(rs.getString("identificadorContacto"));
            rows[3] = aes.descifrar(rs.getString("numMensajesTotales"));
            rows[4] = aes.descifrar(rs.getString("numMensajesSinLeer"));
            rows[5] = aes.descifrar(rs.getString("estaBloqueado"));
            rows[6] = aes.descifrar(rs.getString("secreto"));
            res.add(rows);
        }
    } catch (SQLException ex) {
        mr.addLogLine("BD: (ERROR) [descifraContacto] Error al descifrar un contacto.\n"+
            ex.toString(), true);
    }
    return res;
}
```

Figura A.11: Método para la obtención y descifrado de un contacto de la base de datos.

A.8.3. Inserta mensaje

```

public void insertaMensaje( int idContacto, String texto , String fecha, boolean enviado)
    throws SQLException{
    String query = "INSERT INTO mensajes (idContacto,textoCifrado,fecha,enviado)
        VALUES (?,?,,?)";
    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, idContacto);
        texto = aes.cifrar(texto);
        pstmt.setString(2, texto);
        fecha = aes.cifrar(fecha);
        pstmt.setString(3, fecha);
        pstmt.setString(4, aes.cifrar(enviado+""));
        pstmt.executeUpdate();
        pstmt.close();
        incrementaNumMensajesTotalesContacto(idContacto);
        mr.addLogLine("BD: Mensaje insertado correctamente", false);
    } catch (SQLException e) {
        mr.addLogLine("BD: (ERROR) [insertaMensaje] Error al insertar un mensaje.\n"+
            e.toString(), true);
        throw new SQLException();
    }
}

```

Figura A.12: Método para el cifrado y la inserción de un mensaje en la base de datos.

A.8.4. Obtén mensaje

```

public List<String[]> descifraRowsMensajes(ResultSet rs){
    List<String[]> res = new ArrayList<>();
    try {
        int nColumns = rs.getMetaData().getColumnCount();
        while(rs.next()){
            String[] rows = new String[nColumns];
            rows[0] = aes.descifrar(rs.getString("apodo"));
            rows[1] = aes.descifrar(rs.getString("textoCifrado"));
            rows[2] = aes.descifrar(rs.getString("fecha"));
            rows[3] = aes.descifrar(rs.getString("enviado"));
            res.add(rows);
        }
    } catch (SQLException ex) {
        mr.addLogLine("BD: (ERROR) Error al tranformar el resultado de la query a List ",
            true);
    }
    return res;
}

```

Figura A.13: Método para la obtención y descifrado de un mensaje de la base de datos.

A.8.5. Inserción de un mensaje temporal

Los mensajes temporales se encuentran por defecto cifrados en memoria principal, por lo que se almacenan directamente, sin la necesidad de volverlos a cifrar.

```
public void insertaMensajeTemp(int idContacto, String mensajeJSONCifrado)
throws SQLException{
    String query = "INSERT INTO mensajesTemp (mensajeJSONCifrado,idContacto)
VALUES (?,?)";
    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setString(1, mensajeJSONCifrado);
        pstmt.setInt(2, idContacto);
        pstmt.executeUpdate();
        pstmt.close();
        mr.addLogLine("BD: Mensaje temporal insertado correctamente", false);
    } catch (SQLException e) {
        mr.addLogLine("BD: (ERROR) Error al insertar un mensaje temporal\n"+
e.toString(), true);
        throw new SQLException();
    }
}
```

Figura A.14: Método la inserción de un mensaje temporal (ya cifrado) en la base de datos.

A.8.6. Obtén mensajes temporales

Debido a que los mensajes temporales se encuentran cifrados en memoria principal, solamente se procede a su descifrado cuando van a ser enviados, siendo posteriormente eliminados tras el envío del mensaje.

```
public List<String> obtenMensajesTemp(int idContacto){
    List<String> res = new ArrayList<>();
    mr.addLogLine("DB: Obteniendo obtenMensajesTemp", false);
    String sql = "SELECT mensajeJSONCifrado FROM mensajesTemp where idContacto = ? ";
    try (PreparedStatement pstmt = con.prepareStatement(sql))
    {
        pstmt.setInt(1, idContacto);
        ResultSet rs = pstmt.executeQuery();
        while(rs.next()){
            res.add(rs.getString("mensajeJSONCifrado"));
        }
        pstmt.close();
    } catch (SQLException ex) {
        mr.addLogLine("BD: (ERROR) Error al listar los mensjaes temporales por la id: "+
idContacto+" .\n"+ex.toString(), true);
    }
    return res;
}
```

Figura A.15: Método para la obtención mensajes temporales de la base de datos.

A.8.7. Inserta identificador bloqueado

```

public void insertaBloqueado(String identificadorContacto)
throws SQLException{
    String query = "INSERT INTO bloqueados (identificadorContacto) VALUES (?);";
    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        identificadorContacto=aes.cifrar(identificadorContacto);
        pstmt.setString(1, identificadorContacto);
        pstmt.executeUpdate();
        pstmt.close();
        mr.addLogLine("BD: Nuevo usuario bloqueado insertado correctamente", false);
    } catch (SQLException e) {
        mr.addLogLine("BD: (ERROR) Error al añadir un usuario a bloqueados", true);
        throw new SQLException();
    }
}

```

Figura A.16: Método para el cifrado e inserción de un identificador, que ha sido bloqueado, en la base de datos.

A.8.8. Obtén identificadoras bloqueados

```

public List<String[]> obtenContactosBloqueados(){
    List<String[]> res = new ArrayList<>();
    String sql = "SELECT identificadorContacto FROM bloqueados";
    try (Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(sql)){
        while(rs.next()){
            String[] rows = new String[rs.getMetaData().getColumnCount()];
            rows[0] = aes.descifrar(rs.getString("identificadorContacto"));
            res.add(rows);
        }
        stmt.close();
    } catch (SQLException e) {
        mr.addLogLine("BD: (ERROR) [obtenContactosBloqueados] Error al listar
        los usuarios de la tabla.\n"+
        e.toString(), true);
    }
    return res;
}

```

Figura A.17: Método para la obtención y descifrado de identificadores, que han sido bloqueado, de la base de datos.