



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Securización de una aplicación Android para geolocalización y comunicaciones en despliegues de Infantería de Marina

Grado en Ingeniería Mecánica

ALUMNO: Ignacio Bresca Gómez-Martelo

DIRECTORES: Belén Barragáns Martínez
Pablo Sendín Raña

CURSO ACADÉMICO: 2020-2021

UniversidadeVigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Securización de una aplicación Android para geolocalización y comunicaciones en despliegues de Infantería de Marina

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universida_{de}Vigo

RESUMEN

Resulta fundamental para el éxito de cualquier misión y, en particular, en un despliegue de Infantería de Marina disponer de buenos canales de comunicación entre los participantes de la misión y el puesto de mando y control. En este marco, en el curso 2019-2020 se desarrolló la aplicación ATC (dentro de un TFG) para proporcionar servicios de geolocalización y comunicaciones multimedia (chat, audio y video) a los participantes de la misión. Dicha aplicación se desarrolló en el lenguaje de programación Kotlin para el sistema operativo Android.

Es asimismo muy relevante proporcionar seguridad en dichas comunicaciones, carencia detectada en ATC. En consecuencia, con la finalidad de solventar estas limitaciones, se plantea el TFG. Este trabajo resuelve tanto el problema de ausencia de sistema de autenticación de usuarios como el problema de envío de mensajes en claro, aplicando esquemas de cifrado simétrico y asimétrico en las soluciones respectivas.

En el último capítulo de esta memoria se validan con éxito las implementaciones desarrolladas, así como se proporciona una prueba en escenario real de la aplicación ATC en su versión securizada.

PALABRAS CLAVE

Seguridad, encriptar, ATC, Android, Kotlin

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis tutores Doña Belén Barragáns Martínez y Don Pablo Sendín Raña por la ayuda prestada para realizar el trabajo, sus incansables ganas de mejorar y el seguimiento diario en los avances de este trabajo. Toda esa ayuda ha sido imprescindible para cumplir con los objetivos propuestos inicialmente.

En segundo lugar, agradezco a mis compañeros Álvaro Zumalacárregui, Guillermo Maza y Aurelio Fernández por la ayuda prestada para la validación de las funciones de la aplicación ya que, sin ellos, hubiese llevado mas tiempo y trabajo conseguirlo. Sin embargo, ellos aceptaron sin dudar prestarme su tiempo para lo que necesitaba.

Y tercero, y puede que más importante, quiero agradecer a mi familia, especialmente a mi madre, por su apoyo incondicional a cumplir con todo aquello que me propongo; a mis amigos, que siempre han estado conmigo para subirme la moral en los momentos en los que parece más difícil seguir avanzando, resaltando sobre todo a Guillermo Ferraz, Víctor Rama y Carlos Hertfelder y, por último, a Macarena Moreno que es la persona que más me empuja hacia adelante en cualquier situación de la vida y cree más en mis capacidades que yo mismo.

ÍNDICE

Índice	1
Índice de Figuras	3
1 Introducción y objetivos	5
1.1 Motivación del TFG	5
1.2 Objetivos del TFG	6
1.3 Organización de la memoria	6
2 Estado del arte	7
2.1 Revisión de conceptos de ciberseguridad	7
2.2 Aplicaciones similares a ATC	8
2.2.1 En el ámbito militar	8
2.2.2 Ámbito civil	9
2.3 Antecedentes del TFG previamente realizado	10
2.4 Revisión de tecnologías relacionadas con el TFG	12
2.4.1 Sistema Operativo Android	12
2.4.2 Lenguaje de programación elegido	12
2.4.3 IDE empleado	13
2.5 Revisión de algoritmos de cifrado	14
2.5.1 Cifrado de clave simétrica	15
2.5.2 Cifrado de clave asimétrica	16
3 Desarrollo del TFG	19
3.1 Problemas de seguridad de ATC. Estudio de posibles soluciones	19
3.1.1 Carencia de servicio de autenticación de usuarios	19
3.1.2 Envío de mensajes en claro entre participantes	19
3.2 Implementación de solución de autenticación de usuarios	20
3.3 Implementación de solución de implementación de mensajes	25
4 Validación del TFG	31
4.1 Validación del sistema de autenticación implementado.	31
4.2 Validación de la encriptación de mensajes	34
4.3 Aplicación de ATC en un escenario real	34
4.3.1 Recursos materiales	34
4.3.2 Despliegue en la ENM	36
4.3.3 Comprobación de uso de la clave de operación	37
5 Conclusiones y líneas futuras	39

5.1 Conclusiones	39
5.2 Líneas futuras	39
6 Bibliografía.....	41
Anexo I: Código de la aplicación	45

ÍNDICE DE FIGURAS

Figura 2-1 Usuario empleando ATAK en un despliegue [7]	9
Figura 2-2 Visualización de la aplicación CivTAK en uso [10]	10
Figura 2-3 Diagrama de flujo del funcionamiento de la aplicación [1].	11
Figura 2-4 Sistemas operativos móviles más usados en 2020 [12]	12
Figura 2-5 Interfaz de Android Studio	14
Figura 2-6 Icono para la modificación del SDK de Android Studio	14
Figura 2-7 Secuencia de un cifrado simétrico [18]	15
Figura 2-8 Secuencia de un cifrado asimétrico [21]	16
Figura 3-1 Desplegable <i>Show hidden tabs</i>	20
Figura 3-2 Características del layout para escribir la contraseña	21
Figura 3-3 Pantalla de inicio de la aplicación ATC	21
Figura 3-4 Modificaciones de la clase <i>MainActivity</i>	22
Figura 3-5 Funciones encargadas de mantener fija la clave en 32 caracteres	22
Figura 3-6 Funciones de encriptado y desencriptado del cifrado simétrico	23
Figura 3-7 Cifrado del mensaje broadcast	23
Figura 3-8 Proceso que se realiza para desencriptar el mensaje broadcast recibido	24
Figura 3-9 <i>Intent</i> del ID y la clave de acceso a la aplicación	24
Figura 3-10 Tipo de cifrado empleado (RSA)	25
Figura 3-11 Función que genera las claves pública y privada	25
Figura 3-12 Función para encriptar el mensaje enviado	26
Figura 3-13 Función para desencriptar el mensaje recibido	26
Figura 3-14 Función que se encarga de importar las claves	26
Figura 3-15 Clase <i>Participante</i> modificada para el cifrado asimétrico	27
Figura 3-16 Contenido del mensaje broadcast	27
Figura 3-17 Estructura del mensaje broadcast	27
Figura 3-18 Análisis del mensaje broadcast recibido	28
Figura 3-19 Información acerca del participante seleccionado	28
Figura 3-20 Obtención de los parámetros <i>intent</i> mediante <i>importKeys</i>	28
Figura 3-21 Función que permite enviar mensajes cifrados a otros usuarios	29
Figura 3-22 Función encargada de desencriptar el mensaje recibido	29
Figura 3-23 Ejemplo de imagen del chat desde el dispositivo de Alice	30
Figura 4-1 Interfaz de inicio de Wireshark	32
Figura 4-2 Lista de mensajes transmitidos por dispositivos conectados a la red Wi-Fi	32

Figura 4-3 Información de un mensaje broadcast enviado por un usuario que utiliza la aplicación sin cifrado simétrico	33
Figura 4-4 Información de un mensaje broadcast enviado por un usuario que utiliza la aplicación con cifrado simétrico.....	33
Figura 4-5 Mensaje transmitido por ATC encriptado mediante el cifrado asimétrico.....	34
Figura 4-6 Router LINKSYS WRT 1900 ACS [27]	35
Figura 4-7 Xiaomi Mi A2 Lite [28].....	36
Figura 4-8 Distancia a la que se pueden posicionar los usuarios para estar conectados a la red	36
Figura 4-9 Demostración de la seguridad de la aplicación en cuanto al acceso.....	37

1 INTRODUCCIÓN Y OBJETIVOS

En este capítulo, se describe la motivación de este Trabajo de Fin de Grado, los objetivos a perseguir durante su realización y la forma en la que estará organizada esta memoria.

1.1 Motivación del TFG

Actualmente existen multitud de conflictos armados y misiones de paz. Las Fuerzas Armadas Españolas están presentes en 16 misiones de paz y, concretamente, la Infantería de Marina de la Armada Española ha participado en 7 en los últimos años. En todos ellos, las comunicaciones forman una parte fundamental e indispensable para el cumplimiento de la misión. Dentro de la línea de trabajo de la comunicación, el año pasado se desarrolló un Trabajo de Fin de Grado, que permite a los infantes de marina una comunicación instantánea en el campo de batalla tanto por mensajería como por vídeo, además, permite la geolocalización en tiempo real de todos los participantes del equipo. Este trabajo de fin de grado se denominó *“Diseño y desarrollo de una aplicación en Android para geolocalización y comunicaciones en despliegues de Infantería de Marina utilizando una red local”* [1] y fue llevado a cabo por el teniente Rández Aguillo. El desarrollo de una aplicación como esta aporta una ventaja que tiene mucho valor para el mando y control del equipo debido a que pueden mantener la comunicación en remoto, mediante una red Wi-Fi y el alcance máximo al que podrían establecer esa comunicación quedará determinado por el alcance de esa red. La aplicación fruto de dicho TFG se denomina ATC (Aplicación Táctica de Combate).

Como todo lo que tiene que ver con la manipulación de la información, las comunicaciones están en continuo desarrollo. Por un lado, este desarrollo se aplica tanto al establecimiento de comunicaciones, como a la protección la información transmitida. Por otro lado, ese desarrollo también hace referencia a las formas ilegales de obtener información ajena, lo que puede provocar serios problemas. Estos dos conceptos van ligados uno al otro, una vez se establece un nivel de seguridad para evitar que alguien pueda acceder a la información salvo su emisor y destinatario, los hackers buscan saltar esos sistemas de seguridad. Cuando estos sistemas se detectan como vulnerables, las empresas encargadas de la seguridad actualizan, mejoran o desarrollan nuevos mecanismos de seguridad resistentes a estos ataques. Ante este hecho, los piratas informáticos buscan nuevas formas de acceder a la información oculta. Se trata de un proceso repetitivo y por es eso que las comunicaciones están en continuo desarrollo.

En el aspecto de las comunicaciones en combate, es tan importante establecerlas y mantener la comunicación entre aliados; como su protección, es decir, que no las perturben (introduciendo ruido o alterando su contenido) y que no puedan acceder a ellas.

1.2 Objetivos del TFG

El principal objetivo que se desea alcanzar en este TFG es añadir una capa de seguridad a la aplicación ATC, previamente desarrollada, que permita una comunicación segura en despliegues de Infantería de Marina. Para conseguirlo se pretende:

- Asegurar que solo pueden hacer uso de la aplicación los participantes autorizados.
- Ocultar la información que se transfiere entre los participantes.

Un objetivo secundario es la validación de la aplicación en un escenario lo más realista posible.

1.3 Organización de la memoria

Esta memoria está dividida en seis apartados. El primero, el apartado actual, introduce el tema a tratar explicando lo que ha motivado la realización del trabajo y los objetivos que se buscan alcanzar con este. El segundo capítulo (estado del arte) realiza una revisión de la ciberseguridad y también de aplicaciones similares. Resume el trabajo previo que existe hasta ahora y las herramientas que se van a utilizar en el desarrollo de nuevas funcionalidades de seguridad. En el tercero, se explica el desarrollo de las mejoras introducidas en la aplicación aportando además imágenes para ayudar a comprender al lector la secuencia llevada a cabo. La validación de las mejoras implementadas se expone en el cuarto apartado. Por último, el quinto apartado presenta las conclusiones y las líneas futuras, en él se propondrán futuras funcionalidades y ampliaciones para la mejora de esta aplicación.

2 ESTADO DEL ARTE

Este segundo capítulo, comienza con un repaso de los conceptos generales de la ciberseguridad. A continuación, se revisan aplicaciones similares que existen en la actualidad, tanto civiles como militares. Se explicará en qué consistió el trabajo previo y hasta donde se llegó en el desarrollo de la aplicación. Por último, se revisarán soluciones para el cifrado extremo a extremo en las comunicaciones como aquellas propuestas para abordar el problema de autenticación previa a su uso, debido a que son las funcionalidades que deseamos implementar en este TFG.

2.1 Revisión de conceptos de ciberseguridad

La ciberseguridad [2] engloba todos aquellos conceptos, políticas, métodos, acciones y tecnologías cuya función es proteger la intimidad de los usuarios que utilizan la red. Su finalidad es garantizar un determinado nivel de seguridad ante cualquier adversidad que presente el ciberentorno. Las propiedades de la ciberseguridad a la hora de transmitir información incluyen, al menos, las siguientes propiedades:

- Confidencialidad: Solo los usuarios que se intercambian mensajes tienen acceso a la información transmitida.
- Integridad: El contenido del mensaje no debe poder ser alterado por terceras personas que no interactúen en la transmisión de datos.
- Autenticidad: Cada uno de los usuarios deben poder confirmar la identidad del resto de participantes.
- No repudio: Una vez que se ha enviado un mensaje, la persona que lo ha transmitido no puede decir que no ha sido él.
- Disponibilidad: Todos los usuarios deben poder disponer de los recursos necesarios para llevar a cabo la comunicación.

La ciberseguridad se aborda gracias a los denominados mecanismos de gestión de riesgos, estos mecanismos son la identificación de las posibles amenazas, el riesgo y el coste que supondría que accediesen a nuestra información y las medidas a adoptar para reducir o eliminar esa amenaza.

Existen diversos programas maliciosos o malware cuya finalidad es acceder a información ajena o simplemente impedir el uso de las comunicaciones. Los mas comunes se pueden clasificar en:

- Puerta Trasera: Es una entrada secreta que permite al intruso acceder a la aplicación o a la información sin pasar por los procedimientos de autenticación propios de la aplicación. Necesita un programa anfitrión.
- Caballo de Troya: Se trata de una función que en teoría va a ser beneficiosa o útil para la aplicación, sin embargo, esa función crea daños en la aplicación. Necesita programa anfitrión y es el malware más utilizado.

- Bomba lógica: Se activa al hacer una determinada acción común al usar una aplicación. Hasta que no es activado, permanece inactivo sin llamar la atención. También necesita un programa anfitrión.
- Virus: Se ejecuta a la vez que el programa en secreto, ocasiona problemas como borrado de datos, mal funcionamiento del programa o accede a información oculta. La peor característica del virus es que puede autoduplicarse y afectar también a otros programas. Al igual que las anteriores, necesita un programa anfitrión.
- Gusano: Este malware no necesita programa anfitrión y viaja a través de las redes de ordenadores para “infectar” a los ordenadores. Puede modificar datos, borrarlos, consumir recursos como el ancho de banda o acceder a información privilegiada.
- Bacteria: No deteriora el programa o la aplicación en sí, sino que se reproduce de forma exponencial consumiendo cada vez más recursos hasta que bloquea el sistema. No necesita de un programa anfitrión.

Este tipo de programas explicados son los más comunes y los más utilizados, pero existen otros como *Ransomware*, cuya función principal es impedir el acceso a algunos ficheros o *Spyware*, normalmente usado para robar contraseñas o números de tarjetas de crédito.

También existen diversos mecanismos de ataque a las comunicaciones que es de lo que nos vamos a proteger con este trabajo:

- DDoS [3]: *Distributed Denial of Service* es un ataque distribuido a un servidor al que se le envían multitud de peticiones simultáneas para colapsarla.
- Man-in-the-middle [4]: el atacante se sitúa entre los dos usuarios que se comunican interceptando los mensajes enviados.
- Suplantación de identidad.
- Modificación y borrado de datos enviados.

2.2 Aplicaciones similares a ATC.

2.2.1 En el ámbito militar

Dentro del ámbito militar existe una aplicación que ATC tomó como referencia, dicha aplicación se llama ATAK (*Android Tactical Assault Kit*) [5]. Esta aplicación permite al usuario que la emplea una localización precisa de su posición y del entorno que le rodea además de permitirle comunicarse e intercambiar datos con otros usuarios. Sin embargo, el gran problema es que tiene licencia únicamente para empresas de los Estados Unidos además de multitud de requisitos legales por lo que actualmente no puede ser utilizada en nuestro país.

Esta aplicación americana fue desarrollada en el Centro de Producción TAK (*Tactical Assault Kit*) del Ejército del Aire de los Estados Unidos en 2010 [6]. Se ejecuta en el mismo sistema operativo en el que estamos desarrollando nuestra aplicación, en Android. Otra similitud que tienen es que se ejecutan tanto en smartphone como en tablet (véase Figura 2-1) conectadas por redes WiFi, redes tipo mesh o redes móviles.



Figura 2-1 Usuario empleando ATAK en un despliegue [7]

ATAK es una aplicación fácil, intuitiva y segura. Permite, al igual que ATC, varios tipos diferentes de comunicación ya sea por chat o por streaming de audio y vídeo, compartiendo la ubicación del resto de usuarios y dándole a conocer su situación alrededor.

Actualmente, la utilizan alrededor de 40.000 usuarios de las Fuerzas Armadas estadounidenses y ha sido probada también en ayuda a personas que se encuentran en lugares donde han tenido lugar desastres naturales como, por ejemplo, tras el huracán Florence en 2018 [8] en la costa este de Estados Unidos.

Sin embargo, sus desarrolladores no comparten datos de las funcionalidades de seguridad que ATAK emplea para sus comunicaciones.

2.2.2 Ámbito civil

En el ámbito civil existe una aplicación llamada ATAK-CIV (o CivTAK) [9]. Se trata de una aplicación del mismo creador que ATAK, sin embargo, está adaptada al mundo civil por lo que se le eliminaron ciertas funcionalidades únicamente necesarias en el ámbito militar.

Aun así, se trata de una aplicación muy útil para personas aficionadas al senderismo, a la orientación, a la caza, a la pesca, montañistas, socorristas o estudiosos de sitios de vida silvestre (véase Figura 2-2). Estas son algunas de sus características:

- Aporta mapas, puntos de una ruta, dibujos y ubicaciones de interés.
- Soporta imágenes de alta resolución.
- Alcance, rumbo y otras herramientas de medición.
- Chat y transmisión de datos (compartición de archivos, fotos y vídeo).
- Balizas de emergencia.
- Herramientas para ayudar a la evacuación de heridos.
- Capacidad para mostrar modelos geoespaciales 3D.

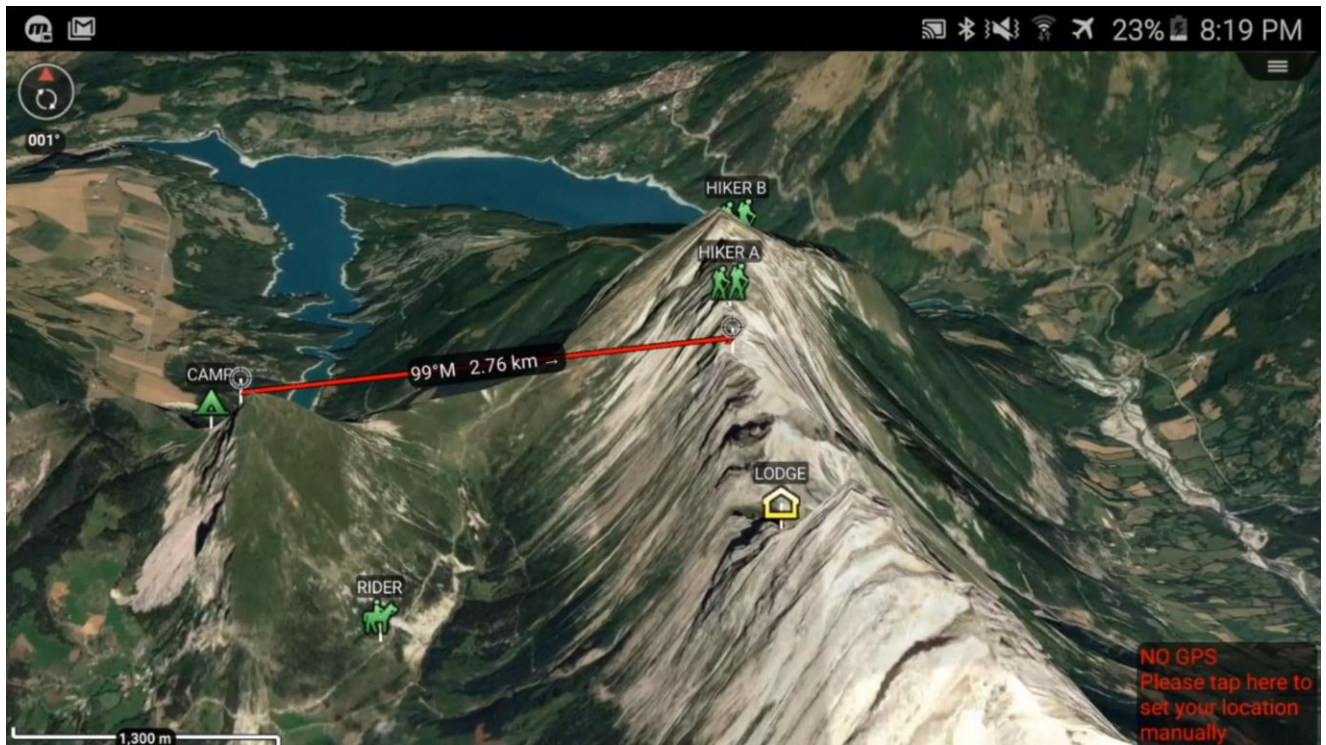


Figura 2-2 Visualización de la aplicación CivTAK en uso [10]

Sin embargo, requiere que el sistema operativo en el que se vaya a instalar cumpla las siguientes características:

- Sistema operativo Android 5.0 o superior.
- Procesador de gráficos que admita GLES 3.0 o superior.
- Hardware equivalente a Samsung S9 o más moderno y en tableta Samsung S2 o más moderno.

2.3 Antecedentes del TFG previamente realizado

Como se adelantó en el primer apartado de esta memoria, este TFG es una continuación de un trabajo que se llevó a cabo el año pasado: *“Diseño y desarrollo de una aplicación en Android para geolocalización y comunicaciones en despliegues de Infantería de Marina utilizando una red local”* [1] elaborado por el teniente Rández Aguillo. Por ello, es oportuno resumir el alcance de dicho trabajo que supone el punto de partida de este TFG.

La aplicación ATC permite compartir la posición GPS en tiempo real de los usuarios que emplean la aplicación simultáneamente, transmitir y recibir información multimedia tanto de audio como de vídeo y un servicio de mensajería instantánea (chat) cuya transmisión se implementa sobre el protocolo de transporte UDP¹, al igual que los mensajes broadcast en los que se comparte la posición GPS de los usuarios. Estas actividades pueden proporcionar una gran ventaja sobre el enemigo si desconoce estas capacidades, o bien, no las puede neutralizar.

Deteniéndonos más en la propia aplicación, una vez es iniciada, pide al usuario escribir un nombre antes de pulsar en el botón de registro. Una vez han accedido a la aplicación los usuarios y se encuentran conectados a la red local, verán en el mapa su posición y la de sus compañeros.

¹ UDP es un protocolo de transporte de mensajes caracterizado por que su transmisión de datos es muy rápida pero a cambio ofrece menores garantías de que no se pierdan paquetes en el proceso que otros protocolos como TCP.

Si presionan sobre las posiciones en las que se encuentran los demás participantes, tienen la opción de enviar un mensaje de texto pulsando en el botón *chat*, o bien, mediante el botón *streaming* podrán consumir video y audio con el usuario elegido.

Por último, si alguno de los usuarios se desconecta de la red, o sale de la aplicación, la aplicación eliminará el marcador de posición de dicho usuario haciendo que éste desaparezca para los demás participantes.

La Figura 2-3 presenta el diagrama de flujo que describe el funcionamiento de la aplicación.

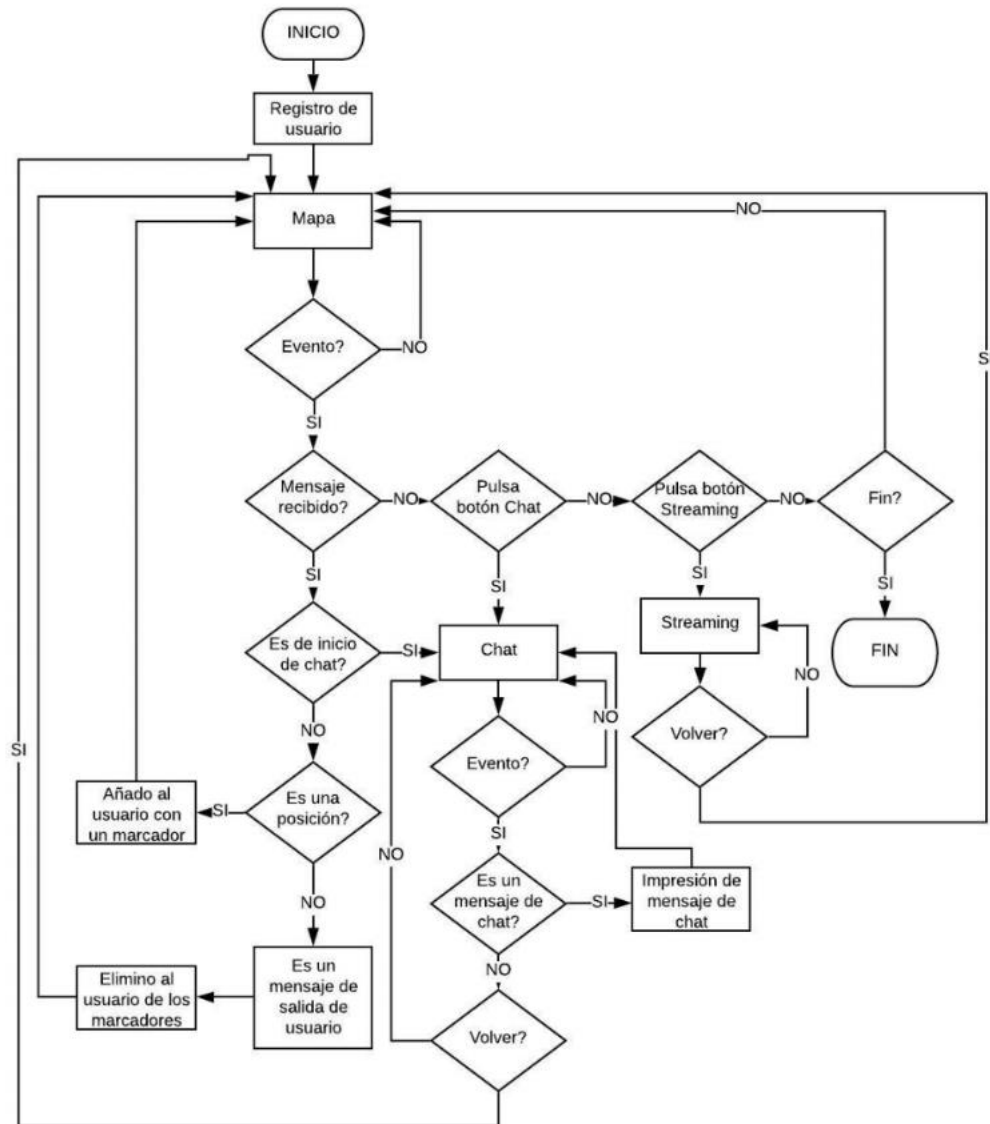


Figura 2-3 Diagrama de flujo del funcionamiento de la aplicación [1].

Una de las carencias que tiene la aplicación, y que motiva el planteamiento de este TFG, es que cualquier persona no autorizada que disponga de la aplicación y tenga acceso a la red local puede acceder a ella y, por consiguiente, acceder a todas las posiciones de los usuarios. Otra razón es que la mensajería instantánea no va cifrada, por lo que cualquier persona que logre captar las comunicaciones obtendrá los mensajes transmitidos por los usuarios pudiendo provocar graves problemas al equipo de Infantería de Marina que lo esté usando.

2.4 Revisión de tecnologías relacionadas con el TFG

En esta sección, se repasarán brevemente las tecnologías empleadas en este TFG, determinadas en gran medida por el trabajo previo que constituye el punto de inicio de este trabajo.

2.4.1 Sistema Operativo Android

El sistema operativo en el que se va a realizar el trabajo es Android debido a que la aplicación hasta ahora, ha sido desarrollada en este sistema operativo. Android es el sistema operativo más utilizado mundialmente frente a otros como iOS, Samsung o Windows. El porcentaje de móviles o tablets que utilizan sistema operativo Android asciende hasta el 71,42% y su perseguidor más cercano es iOS con un 27,95% según datos de diciembre del año 2020 [11] (véase Figura 2-4). Que sea el más empleado implica consigo una mayor calidad, ya que, cuanto mayor sea la demanda y el trabajo en su desarrollo, mayor será la competencia por el mejor producto, en este caso, al tratarse del sistema operativo más utilizado, también se trata del más perfeccionado.

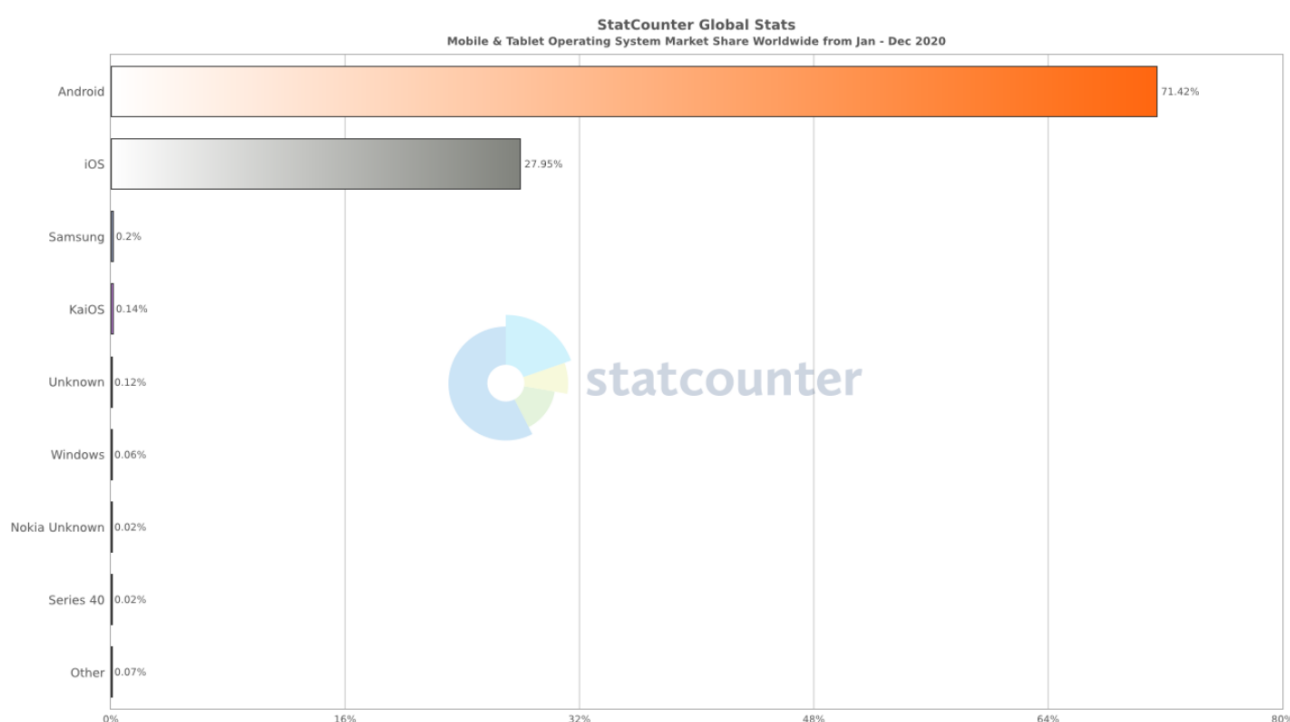


Figura 2-4 Sistemas operativos móviles más usados en 2020 [12]

También es importante la capacidad de difusión y uso. Cuanta más gente utilice este sistema operativo, más personas podrán acceder a los servicios que ofrezca la aplicación tanto si se trata de una aplicación pública o privada.

Por último, se trata de un sistema de código abierto por lo que gran cantidad de desarrolladores trabajan con él y lo mejoran continuamente. Actualmente está pensado para utilizarse en aplicaciones de teléfonos móviles o tablets cuyo uso es mediante pantalla táctil. Otros sistemas operativos funcionan con código cerrado en el que solo el fabricante puede hacer modificaciones.

Por las ventajas expuestas de Android y por ser el sistema operativo utilizado por ATC no se contempla su desarrollo en otro sistema operativo.

2.4.2 Lenguaje de programación elegido

El lenguaje de programación que se va a emplear para el desarrollo de este trabajo es Kotlin y, al igual que en el apartado anterior, es debido a que el TFG anterior trabajo fue realizado usando dicho

lenguaje. Kotlin es un lenguaje de programación moderno (creado en 2011) que utilizan mas del 60% de los desarrolladores de Android.

Fue creado y desarrollado por la empresa JetBrains y pensado para funcionar con la Máquina Virtual de Java (JVM) y Android [13]. El uso de Kotlin ha crecido mucho durante los últimos años gracias al apoyo de Google a la hora de programar aplicaciones, en detrimento de Java, debido a que posee las siguientes características [14]:

- Es interoperable al 100% con Java ya que se pueden usar a la vez en el mismo proyecto.
- Utiliza un lenguaje conciso mediante el cual es capaz de expresar las mismas ideas en menos líneas que Java.
- Se trata de un código más seguro. Kotlin tiene un 20% menos de probabilidades de fallar debido a que no tiene excepciones nulas.
- Posee unas corrutinas, o patrones de diseño, que simplifican el código y agilizan las tareas más comunes, lo que provoca una mayor productividad a la hora de emplearlo.

En definitiva, Kotlin es un lenguaje de programación muy útil, más sencillo e intuitivo que otros con similares prestaciones, que está en auge y diseñado para ser ejecutado en el sistema operativo más utilizado en la actualidad. Por todo ello, y por ser el originalmente elegido para desarrollar ATC, es el lenguaje de programación elegido para el desarrollo del trabajo.

2.4.3 IDE empleado

El IDE [15] (*Integrated Development Environment*) es un programa que permite el diseño de aplicaciones combinando herramientas del desarrollador comunes en una sola interfaz gráfica. Se trata, además, de un sistema que permite programar aplicaciones de forma rápida.

Para la realización del trabajo, se barajaron dos opciones de IDE: Android Studio [16] e IntelliJ Idea [17]. Ambos se utilizan para desarrollar aplicaciones y tienen muchas características en común.

Por un lado, Android Studio está especializado en crear aplicaciones en Android y proporciona funciones que interaccionan con teléfonos móviles. Es por esto que muchos desarrolladores la utilizan para crear aplicaciones de uso público. Sin embargo, es poco útil para otros sistemas operativos.

Por el otro lado, IntelliJ IDEA tiene un entorno de desarrollo más general, lo que le hace más útil en aplicaciones que se vayan a emplear en distintos sistemas operativos. Sin embargo, carece de algunos detalles mas especializados para aplicaciones Android.

Una vez expuestos ambos entornos de desarrollo, para el trabajo se ha decidido utilizar Android Studio (véase Figura 2-5) debido a que, la aplicación a desarrollar se ejecutará en el sistema operativo Android y este IDE seleccionado nos aporta mayores funciones que IntelliJ IDEA.

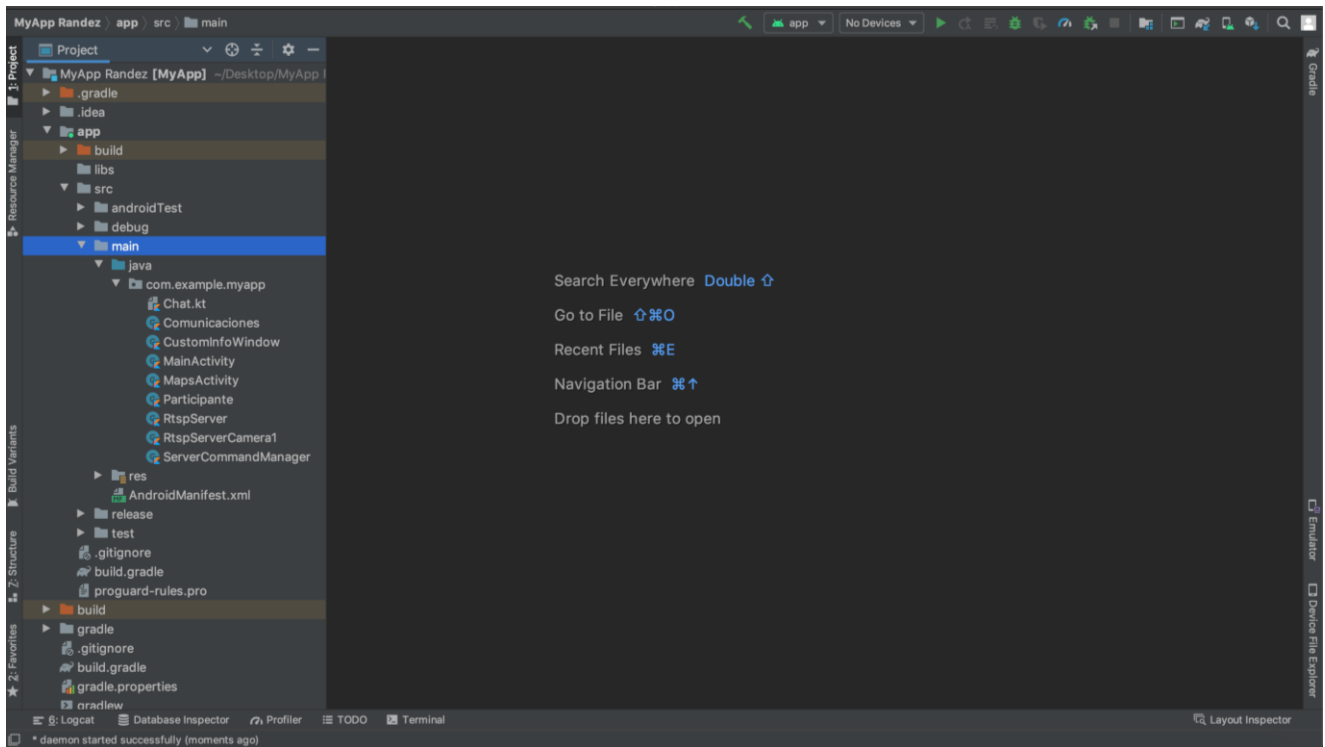


Figura 2-5 Interfaz de Android Studio

Además, para continuar con el trabajo previamente realizado, es importante resaltar que hay que configurar el SDK (*Software Development Kit*) de Android (véase Figura 2-6), en Android 8.0 (Oreo) ya que no es la opción por defecto al descargar Android Studio. Si no se lleva a cabo esta modificación, el código dará multitud de errores y no podrá ser ejecutado.

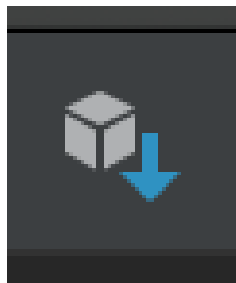


Figura 2-6 Icono para la modificación del SDK de Android Studio

2.5 Revisión de algoritmos de cifrado

Actualmente, la mayoría de las aplicaciones que proporcionan cualquier funcionalidad de mensajería implementa algún tipo de cifrado de los mensajes, ya que, si no fuese así, cualquiera podría leer conversaciones ajenas. De este modo, solo las personas emisoras y receptoras de los mensajes son capaces de verlos. Hoy en día, existen multitud de piratas informáticos que buscan información privada de manera ilegal para usarla en contra de la persona a quién pertenece dicha información.

En el caso particular de la aplicación objeto de este TFG, si no estuviesen cifradas las comunicaciones, terceros podrían acceder al contenido de los mensajes sin ningún tipo de problema. Sin embargo, si el mensaje está cifrado, cuando alguien pretenda intervenir los mensajes, estos serán ilegibles si no conoce la clave. Posiblemente se pueda conocer fácilmente el tipo de cifrado, pero no sirve de nada. La única forma de hacerlo legible es con la clave de descryptación, sin ella, descifrar una conversación será sumamente difícil.

Ante estas amenazas se desarrollaron distintos tipos de cifrado. El cifrado produce una cadena de bits aleatoria para un usuario externo. Este cifrado es reversible mediante la clave de cifrado. Existen dos esquemas principales de cifrado que revisamos en los siguientes apartados.

2.5.1 Cifrado de clave simétrica

La Figura 2-7 representa visualmente la secuencia que se emplea en un esquema de cifrado de clave simétrica.



Figura 2-7 Secuencia de un cifrado simétrico [18]

Este tipo de cifrado se caracteriza por que todos los usuarios utilizan la misma clave secreta y además la clave para cifrar y descifrar es la misma. Este tipo de cifrado permite cumplir con las propiedades de ciberseguridad de confidencialidad e integridad. Dentro del cifrado simétrico destacan dos diferentes de entre todos los existentes:

2.5.1.1 One time-pad

One-time pad [19] se trata de una técnica de encriptación imposible de descifrar pero que requiere que la clave sea compartida previamente. Si no se compartiera la clave, solo quién encripta la transmisión es capaz de descifrarla. Para que cumplir su característica de indescifrable necesita cumplir unos requisitos como son:

1. Ser una clave completamente aleatoria.
2. Que la clave sea como mínimo igual de larga que el texto a encriptar.
3. Que nunca sea reutilizada total o parcialmente una clave.

Este tipo de encriptación ha sido muy utilizada para comunicaciones militares o diplomáticas, sin embargo, la dificultad de compartir la clave de forma discreta ha hecho que no sea un cifrado tan utilizado como otros.

Como ejemplo de uso de este tipo de encriptación, indicar que se utilizó en la Guerra Fría para el uso del “teléfono rojo”. El “teléfono rojo” fue el enlace que permitió la comunicación entre el Presidente de Estados Unidos y la Unión Soviética y fue un hecho muy importante ya que gracias a este medio de comunicación se evitó el comienzo de una guerra nuclear.

2.5.1.2 AES

AES (*Advanced Encryption Standard*) [20] constituye uno de los sistemas de cifrado más utilizados y con mayor éxito de la actualidad. Utiliza un esquema de cifrado por bloques de 128 bits en cuatro matrices de 16 bytes.

Cada byte de datos es sustituido por otro mediante una tabla predeterminada para, a continuación, coger cada matriz y moverla de la siguiente manera: los bytes de la segunda fila se mueven un espacio a la izquierda, los bytes en la tercera fila se mueven dos espacios a la izquierda y los bytes de la cuarta se mueven tres hacia la izquierda para finalmente mezclar las columnas y añadir la clave inicial al conjunto y vuelta a empezar.

Esto genera un texto cifrado que nada tiene que ver con el original y que además visualmente parece carecer de sentido. Para descifrarlo se requiere invertir los pasos conociendo previamente dicha clave.

En cuanto al cifrado AES, el de 128 bits es el que se utiliza actualmente. Sin embargo, algunas empresas y gobiernos utilizan cifrado AES 256. Este tipo de cifrado es indescifrable ya que con los dispositivos que poseemos actualmente, descifrar una sola clave encriptada mediante el AES 256 requeriría el doble de tiempo que la edad del universo.

2.5.2 Cifrado de clave asimétrica

La Figura 2-8 representa la secuencia seguida en un esquema de cifrado de clave asimétrica.

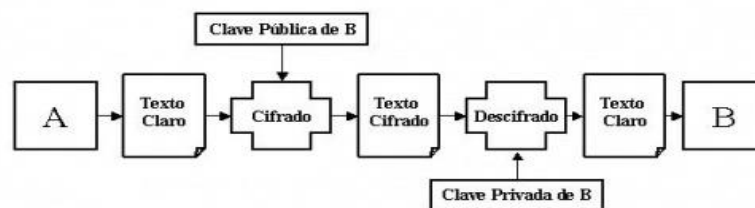


Figura 2-8 Secuencia de un cifrado asimétrico [21]

El cifrado de clave asimétrica se denomina también cifrado de clave pública. En este tipo de cifrado, cada usuario poseerá dos claves, una pública (con la que se cifran los datos) y una privada (que sirve para descifrar los datos), la privada se la queda cada usuario y la pública la envía a los demás para que puedan cifrar las comunicaciones que le envíen a él.

El modo de empleo de la clave asimétrica es el siguiente: El usuario A quiere enviar unos datos al usuario B, para ello, tiene que cifrar esos datos con la clave pública de B, de ese modo únicamente B puede descifrar los datos con su clave privada. Como respuesta, B quiere enviarle otros datos a A por lo que los cifra con la clave pública de A y una vez le llegan los datos cifrados al usuario A, éste los descifra usando su clave privada.

Existen diferentes tipos de algoritmos asimétricos y estos son los más empleados en la actualidad que se describen en los siguientes párrafos.

2.5.2.1 RSA

RSA (*Rivest, Shamir y Adleman*) [22] fue desarrollado en 1979 y es el algoritmo de cifrado asimétrico más utilizado en la actualidad tanto para cifrar datos como para firma digital. Representa los mensajes enviados mediante números, funciona mediante el producto de dos números primos elegidos al azar y de mucho valor (el producto tiene que ser del orden de 10^{300}) y cada vez serán de mayor valor conforme los ordenadores tengan mayor capacidad para los cálculos.

El cifrado RSA seguirá siendo seguro mientras no se invente una manera de descomponer números grandes de forma rápida, por lo que el descifrado de este tipo de cifrados aún no se ha descubierto, es decir, no existe a día de hoy un algoritmo eficiente para resolverlo.

2.5.2.2 DSA

DSA (*Digital Signature Algorithm*) [23] fue diseñado por el NIST (*National Institute for Standards and Technology*) y salió a la luz en 1991. Junto con el anteriormente mencionado RSA, son los dos algoritmos puramente asimétricos. Sin embargo, este algoritmo no es utilizado para cifrar mensajes o datos, sino para firmarlos.

3 DESARROLLO DEL TFG

En este capítulo se describirá el proceso seguido en el desarrollo del TFG. En primer lugar, se diagnostican los problemas de seguridad de ATC y se estudian posibles soluciones. Con los apartados 3.2 y 3.3 se detalla la implementación en Kotlin de las soluciones diseñadas.

3.1 Problemas de seguridad de ATC. Estudio de posibles soluciones

En este apartado se presentan las carencias identificadas en ATC en cuanto a la seguridad en las comunicaciones, así como la propuesta de solución seleccionada para paliar estas limitaciones.

3.1.1 Carencia de servicio de autenticación de usuarios

La aplicación ATC no realiza ningún tipo de autenticación de usuarios, cualquier persona que consiga acceder a un teléfono con la aplicación instalada y conectarse a la misma red local, puede acceder a todas las funcionalidades de ATC. Para evitar que esto ocurra, se decide definir una contraseña de operación que será válida temporalmente mientras dure la misión. Dicha contraseña se solicita al acceder a la aplicación y con ella se encriptan y desencriptan mensajes broadcast. Como esta contraseña de operación es conocida por todos los usuarios de ATC lícitos, las comunicaciones entre ellos iniciales (mensajes broadcast para dar a conocer las posiciones que se ocupan, sus identificadores y sus direcciones IP, además de otros mensajes broadcast para iniciar *chat* o *streaming*) pueden ir cifradas de forma simétrica. El que no posea la contraseña de operación no puede conocer las posiciones de los compañeros del equipo ni acceder a las funcionalidades de *stream* y *chat* con ellos.

3.1.2 Envío de mensajes en claro entre participantes

Además, ATC recibe mensajes en claro entre usuarios, lo que puede suponer un peligro para quien utilice la aplicación porque el enemigo puede interceptar esos mensajes transmitidos. De todos los sistemas analizados se opta por implementar un esquema de un cifrado asimétrico. De este modo, cuando un participante quiera iniciar una conversación de chat con otro participante, solo tendrá que cifrar la clave pública del destinatario para implementar un cifrado asimétrico. Esto es así porque, aunque la aplicación ATC es distribuida y no se dispone de ninguna infraestructura de clave pública (ninguna Autoridad de Certificación), en los mensajes broadcast que dan a conocer las posiciones se incorpora un nuevo campo para propagar entre todos los usuarios lícitos de ATC las claves públicas de los participantes.

3.2 Implementación de solución de autenticación de usuarios

Con el cifrado simétrico se consigue cifrar parte de las comunicaciones como son los mensajes broadcast, a la par que también nos permite impedir el acceso a la aplicación a usuarios ajenos, confundiéndoles y haciéndoles creer que han accedido a la aplicación y su información, pero no es así.

Este cifrado solo emplea una clave, la clave de operación. Ya que no tenemos que distribuir esta clave entre los participantes, porque es una clave conocida previamente por todos los participantes en la operación, su implementación es sencilla.

A continuación, se detallan los pasos seguidos para implementar este tipo de cifrado y sus funcionalidades en la aplicación ATC:

En primer lugar, se diseña el *layout* que va a permitir introducir la clave de operación. Para ello, hay que seleccionar en las pestañas ocultas la opción de Android (véase Figura 3-1) y, una vez ahí, seleccionar dentro de la carpeta *layouts*, el archivo *activity_main.xml*.

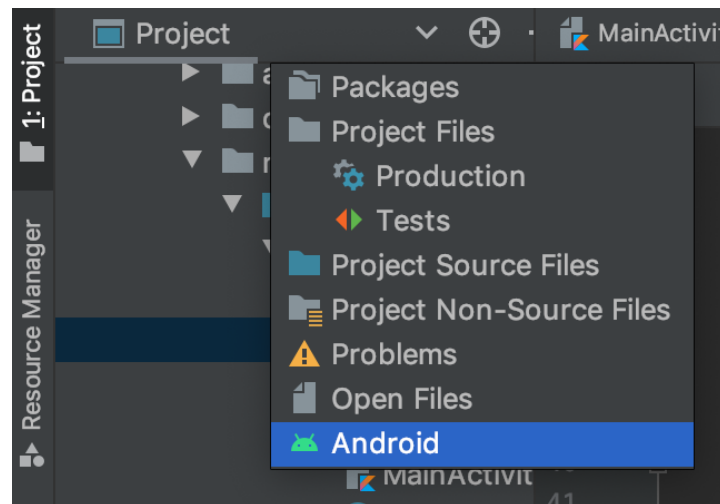


Figura 3-1 Desplegable Show *hidden tabs*

Dentro del *activity_main.xml*, se ha incorporado un campo de tipo *EditText*. Sus principales características son que mediante la funcionalidad *editTextTextPassword* (Figura 3-2) se consigue que, una vez se escriba cada letra, esta se convierta en un punto para que no sea legible el texto escrito. Además, el espacio para escribir la contraseña está indicado mediante el texto: “Escribe la contraseña”. Por último, los *layouts* se utilizan para posicionar e indicar el tamaño que va a tener el espacio en el que escribir la contraseña.

```
26 <EditText
27     android:id="@+id/editTextTextPassword"
28     android:layout_width="0dp"
29     android:layout_height="wrap_content"
30     android:layout_gravity="center"
31     android:layout_marginTop="142dp"
32     android:layout_weight="1"
33     android:background="@android:color/transparent"
34     android:ems="10"
35     android:hint="Escribe la contraseña"
36     android:inputType="textPassword"
37     android:maxLines="1"
38     app:layout_constraintTop_toTopOf="parent"
39     tools:ignore="MissingConstraints"
40     tools:layout_editor_absoluteX="0dp" />
```

Figura 3-2 Características del layout para escribir la contraseña

La pantalla de inicio de la aplicación en el dispositivo móvil queda configurado como se muestra en la Figura 3-3, donde el usuario para acceder se tiene que identificar y después definir la clave compartida por todos los participantes de ese ejercicio o maniobra.

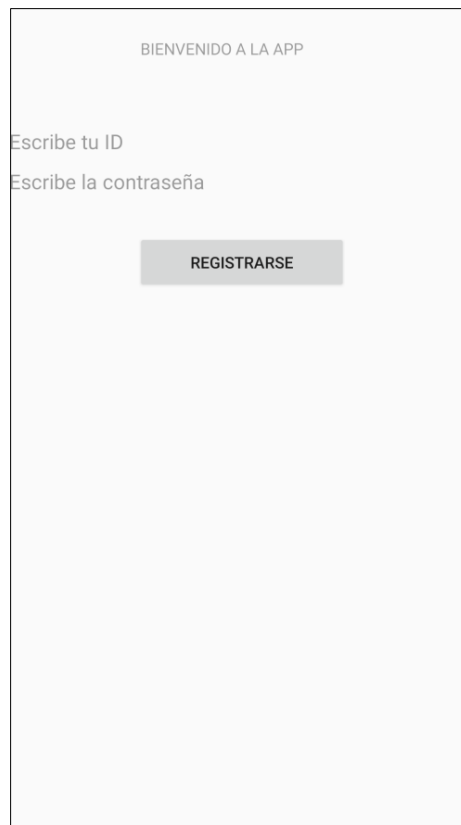


Figura 3-3 Pantalla de inicio de la aplicación ATC

Para la implementación del cifrado simétrico con los requerimientos específicos se ha tenido que modificar el código de las clases *MainActivity* y *MapsActivity*.

En primer lugar, en *MainActivity*, se realizó un pequeño cambio (véase Figura 3-4) para modificar el *intent*² [24] que se le envía a *MapsActivity*. Ahora también incluye la cadena de caracteres que define la clave simétrica para el cifrado de mensajes broadcast.

```

22
23      var mId: String = editTextId.text.toString()
24
25      var claveAcceso: String = editTextTextPassword.text.toString()
26
27
28      val intent = Intent( packageContext: this, MapsActivity::class.java)
29      intent.putExtra( name: "Id", mId)
30      intent.putExtra( name: "claveAcceso", claveAcceso)
31      startActivity(intent)

```

Figura 3-4 Modificaciones de la clase *MainActivity*

Ya en la clase *MapsActivity*, se incluyeron nuevas funciones. Para empezar, se llevó a cabo la creación de las funciones *hashstring*, *string.md5* y *string.sha256* (ver Figura 3-5). La finalidad de estas funciones es que, independientemente del número de caracteres que compongan la clave de acceso (siempre que no supere los 32) se genera un hash de 32 caracteres a partir de la clave simétrica. Este es un requerimiento de las funciones que nos permitirán cifrar de forma simétrica.

```

61
62
63      fun String.md5(): String{
64          return hashString( input: this, algorithm: "MD5")
65      }
66
67      fun String.sha256():String{
68          return hashString( input: this, algorithm: "SHA-256")
69      }
70
71      fun hashString(input: String, algorithm: String): String{
72          return MessageDigest.getInstance(algorithm).digest(input.toByteArray()).fold( initial: "", {str, it -> str + "%02x".format(it)} )

```

Figura 3-5 Funciones encargadas de mantener fija la clave en 32 caracteres

Se incluyen también funciones para el cifrado y descifrado como se puede ver en la Figura 3-6 los mensajes broadcast que se envían desde el dispositivo y, además, también se descifran los mensajes broadcast transmitidos por el resto de usuarios.

Tanto en el cifrado como en el descifrado primero se define método de cifrado, en este caso *AES*, y posteriormente la opción *GCM/NoPadding*. *GCM* [25] (Galois Counter Mode) que es un modo de operación para cifrado en bloque y realiza un algoritmo de cifrado que permite encriptación y autenticación simultáneamente. *NoPadding* indica que no se debe autorrellenar el mensaje.

Mediante el *cipher.init* se define la clave que permitirá el cifrado o descifrado y la función *cipher.doFinal* es la que realiza ese cifrado o descifrado.

En el descifrado aparece un *try and catch* cuya función es que si por cualquier error, no se descifra correctamente, la aplicación no se reinicie o se cierre abruptamente, sino que, únicamente, muestre el error en el terminal.

² Un *intent* sirve para invocar componentes de otras *Activities* en Android

```

72
73 @SuppressWarnings("GetInstance")
74 fun encrypt(strToEncrypt: String, password: String): String {
75     val cipher : Cipher! = Cipher.getInstance( transformation: "AES")
76     val key = SecretKeySpec(password.toByteArray(), algorithm: "AES/GCM/NoPadding")
77
78     cipher.init(Cipher.ENCRYPT_MODE, key)
79     val claveEncriptada : ByteArray! = cipher.doFinal(strToEncrypt.toByteArray())
80
81     return Base64.encodeToString(claveEncriptada, Base64.DEFAULT)
82 }
83 @SuppressWarnings("GetInstance")
84 fun decrypt(strToDecrypt: String, password: String): String {
85     val cipher : Cipher! = Cipher.getInstance( transformation: "AES")
86     val key = SecretKeySpec(password.toByteArray(), algorithm: "AES/GCM/NoPadding")
87
88     cipher.init(Cipher.DECRYPT_MODE, key)
89     try {
90
91         val claveDesencriptada : ByteArray! = cipher.doFinal(Base64.decode(strToDecrypt, Base64.DEFAULT))
92         return String(claveDesencriptada)
93     } catch (e: Exception) {
94     }
95     return ""
96 }
97
98 }

```

Figura 3-6 Funciones de encriptado y desencriptado del cifrado simétrico

A continuación se muestran las modificaciones que se han realizado en el envío del broadcast (véase Figura 3-7). En primer lugar, se ha encriptado llamando a la función *encrypt* anteriormente mostrada. En segundo lugar, se le añaden los caracteres “ATC.” antes de la cadena del broadcast cifrado con la finalidad de localizarlo y diferenciarlo de otros mensajes broadcast de otros dispositivos. Con equivalente procedimiento se cifran los mensajes de inicio de *chat* y de *stream*. Por último, se envía mediante la función *sendUDP*.

```

152 //PASA A LA VARIABLE "MESSAGE" NOMBRE,LAT, LONG E IP PARA EMITIR UN BROADCAST A TRAVÉS DE LA CLASE DE COMUNICACIONES
153 var mensajeBroadcast = String()
154 var pubKeyString :String? = keys!!.publicKeyString
155 mensajeBroadcast = miId+"."+miLatitud+"."+miLongitud+"."+ip+"##"+pubKeyString+":"
156
157 var mensajeBroadcastEncriptado :String = encrypt(mensajeBroadcast, hashString(claveAcceso, algorithm: "MD5"))
158 var mensajeEnviado :String = "ATC."+mensajeBroadcastEncriptado
159
160 udpComm.sendUDP(mensajeEnviado)

```

Figura 3-7 Cifrado del mensaje broadcast

Para realizar el desencriptado del mensaje broadcast (Figura 3-8), se debe seleccionar los primeros 612 caracteres que recibimos e, inmediatamente después, si los cuatro primeros caracteres son “ATC.”, sabemos que se trata de nuestro mensaje y se procede a desencriptar el mensaje mediante la función *decrypt* mostrada anteriormente. La razón por la que se seleccionan solo los primeros 612 caracteres del mensaje recibido quedará explicada más adelante.

```

210
211 fun receiveBroadcast() {
212
213     while (true) {
214         // Recibimos el mensaje de broadcast pero también los mensajes de CHAT
215         var mensajeRecibidoEncriptado :String = udpComm.receiveUDP( broadcast: true)
216         mensajeRecibidoEncriptado = mensajeRecibidoEncriptado.substring(0,612)
217
218         if (mensajeRecibidoEncriptado.substring(0,4) == "ATC.") {
219
220             //Es nuestro mensaje
221             var recibido :String = decrypt(mensajeRecibidoEncriptado.substring(4,
222                 mensajeRecibidoEncriptado.length),
223                 hashString(claveAcceso, algorithm: "MD5"))
224

```

Figura 3-8 Proceso que se realiza para descryptar el mensaje broadcast recibido

En la Figura 3-9 se muestra el código que recoge del intent la clave de operación y se transforma a un *hash* del tamaño exigido para cifrar de forma simétrica. También se puede apreciar en la imagen como mediante *intent* se obtienen las variables *miId* y *claveAcceso*, mencionadas anteriormente, de la clase *MainActivity*.

```

165 //INTENT QUE OBTIENE DEL MAINACTIVITY LA VAR "ID" Y LA CLAVE DEL USUARIO QUE INICIA SESIÓN
166 val objetoIntent: Intent = intent
167
168 miId = objetoIntent.getStringExtra( name: "Id")
169 claveAcceso = objetoIntent.getStringExtra( name: "claveAcceso")
170
171 //CIFRADO DE LA CLAVEACCESO
172 val code :String = encrypt(claveAcceso, hashString(miId, algorithm: "MD5"))
173

```

Figura 3-9 Intent del ID y la clave de acceso a la aplicación

Los principales problemas o dificultades que se experimentan durante el cifrado simétrico se produjeron a la hora de realizar la encriptación y descryptación de los mensajes broadcast:

La aplicación intentaba descryptar con clave de operación todos los mensajes broadcast que recibía: los generados por la aplicación ATC y el resto generados por otras aplicaciones. Al intentar descifrar mensajes broadcast de otras fuentes, la aplicación lanzaba errores. Para solucionarlo, se decidió que al mensaje encriptado se le añadieran los caracteres “ATC.” al inicio de este antes de enviarlo y de esta forma identificar los mensajes que debemos descifrar con la clave de operación. Estos caracteres se adjuntaron al mensaje ya que, de esa manera, la función de descryptación sabrá qué mensajes debe descryptar. Aunque le llegan multitud de mensajes broadcast de otros dispositivos conectados a la red, solo se debe descryptar aquellos que comienzan por esos cuatro caracteres. De este modo, se consigue localizar los mensajes que debe descifrar y no intenta descifrar todos los mensajes pudiendo dar lugar a algún error.

Debido a que el mensaje UDP recibido contiene caracteres adicionales aleatorios (problema arrastrado de la aplicación ATC) la función de descifrado fallaba por ser un mensaje diferente al enviado. Sabiendo el tamaño de los mensajes cifrados (siempre 612 caracteres) se procedió a quedarse con los primeros 612 caracteres (mediante el comando *mensajeRecibido.substring(0,612)*). De este modo el mensaje enviado y recibido coincidían y la función de descifrado tiene éxito.

3.3 Implementación de solución de implementación de mensajes

Para el cifrado asimétrico se crea una nueva clase denominada *KeyGenerator*. Esta clase se utiliza para la generación del par de claves, la función de importar las claves de *String* (cadena de caracteres) a *Key* (clave) y las funciones de cifrado y descifrado asimétrico de mensajes. Este cifrado se emplea para encriptar los mensajes de chat que se envían unos usuarios a otros. El cifrado asimétrico proporciona confidencialidad, integridad, autenticidad y no repudio.

En cuanto al tipo de cifrado, se ha utilizado el cifrado RSA de 2048 bits. Dentro de éste, la opción *ECB/OAEPWithSHA1AndMGF1Padding* donde ECB es un modo de operación de cifrado en bloque y *OAEPWithSHA1AndMGF1Padding* indica que el texto está sujeto al método de codificación *OAEP*. *OAEP* evita posibles vulnerabilidades y no permite que el mensaje sea modificado sin que el destinatario lo detecte durante el descifrado. En este caso, se utilizan 2048 bits, (cuantos más bits, mayor seguridad). Sin embargo, tarda más tiempo en hacer las operaciones matemáticas de encriptado, aunque en el caso de ATC, no afecta a su rendimiento. En la Figura 3-10 se muestran los atributos que definen estos parámetros.

```
companion object {  
    const val CRYPTO_METHOD = "RSA"  
    const val CRYPTO_BITS = 2048  
    private const val OPCION_RSA = "RSA/ECB/OAEPWithSHA1AndMGF1Padding"  
}
```

Figura 3-10 Tipo de cifrado empleado (RSA).

Para la generación del par de claves se utilizó la clase *KeyPairGenerator*, dentro de la función *generateKeyPair* (véase Figura 3-11). Se encarga de inicializar en primer lugar el objeto con el tipo de algoritmo criptográfico elegido (en nuestro caso, RSA) y el tamaño en bits de las claves (2048). El par de claves se guarda en la variable *kp* [26].

```
@Throws(Exception::class)  
public fun generateKeyPair() {  
    //GENERA E INICIALIZA LAS CLAVES PÚBLICA Y PRIVADA  
    var kpg = KeyPairGenerator.getInstance(CRYPTO_METHOD)  
    kpg.initialize(CRYPTO_BITS)  
    //GUARDAMOS LAS CLAVES EN kp  
    var kp = kpg.generateKeyPair()  
}
```

Figura 3-11 Función que genera las claves pública y privada

En la función de encriptación (ver Figura 3-12) se emplea la clase *cipher* que nos proporciona la funcionalidad de cifrado criptográfico, es decir, nos va a permitir cifrar y descifrar con el algoritmo de cifrado elegido. Mediante *cipher.setInstance* indicamos el método de cifrado (RSA) mediante *cipher.init* indicamos que va a cifrar y la clave pública que se utilizará para el cifrado. Con *cipher.doFinal* se cifra el mensaje de texto en claro. El resultado se almacena en la variable *encryptedByte* y, finalmente, la función devuelve esa cadena en formato *Base64*.

```

@Throws(Exception::class)
public fun encrypt(chatMessage: String): String {
    val cipher = Cipher.getInstance(OPTION_RSA)
    cipher.init(Cipher.ENCRYPT_MODE, this.publicKeyEncodedKeySpec)
    encryptedByte = cipher.doFinal(chatMessage.toByteArray())
    return Base64.encodeToString(encryptedByte, Base64.DEFAULT)
}

```

Figura 3-12 Función para encriptar el mensaje enviado

La función de desencriptación (véase Figura 3-13) es similar a la de encriptación. En primer lugar, la clase *cipher* nos proporciona la funcionalidad de cifrado, con *cipher.getInstance* definimos el algoritmo (RSA). *Cipher.init* indica que vamos a descifrar la clave privada que se utilizará para el desencriptado. Con *cipher.doFinal* se descifra el mensaje recibido y el resultado se almacena en la variable *decryptedString*, que es la variable *decryptedByte* pasado a cadena de caracteres.

```

90
91 @Throws(Exception::class)
92 public fun decrypt(mensajeRecibido: String): String {
93
94     val cipher : Cipher! = Cipher.getInstance(OPTION_RSA)
95     cipher?.init(Cipher.DECRYPT_MODE, this.privateKey)
96     decryptedByte = cipher!!.doFinal(Base64.decode(mensajeRecibido, Base64.DEFAULT))
97     decryptedString = String(decryptedByte)
98     return decryptedString!!
99 }
100

```

Figura 3-13 Función para desencriptar el mensaje recibido

Dentro de la clase *KeyGenerator* (detallada en la Figura 3-14) se encuentra, por último, la función *importKeys* cuya finalidad es pasar las variables *publicKey* y *privateKey* de *String* (cadena de caracteres) a *Key* (clave) para que realicen su función de cifrado y descifrado. En esta función, se tuvo un pequeño problema pues la librería *java.util.Base64* no se podía importar simultáneamente con la librería *Android.util.Base64* por lo que se decidió agregar la librería únicamente para esta función de la manera en la que aparece en la imagen.

```

68
69 @Throws(Exception::class)
70 public fun importKeys(pubKey:String?,privKey:String?) {
71
72     val kf: KeyFactory = KeyFactory.getInstance( algorithm: "RSA")
73     val keySpecPKCS8: PKCS8EncodedKeySpec = PKCS8EncodedKeySpec(java.util.Base64.getDecoder().decode(privKey))
74
75     val keySpecX509: X509EncodedKeySpec = X509EncodedKeySpec(java.util.Base64.getDecoder().decode(pubKey))
76
77     publicKey = kf.generatePublic(keySpecX509) as RSAPublicKey
78     privateKey = kf.generatePrivate(keySpecPKCS8)
79 }
80

```

Figura 3-14 Función que se encarga de importar las claves

Además de crear la clase *KeyGenerator*, también se modificó la clase *MapsActivity* puesto que es la que hace de nexo de unión entre las clases *Chat*, *Participante* y *KeyGenerator*, necesarias para desarrollar la comunicación entre los diferentes usuarios que accedan a la aplicación y que también

sufrieron algunas modificaciones para conseguir un funcionamiento seguro de la mensajería instantánea encriptada.

En la clase *Participante* únicamente se añadió, a las variables creadas anteriormente, una nueva variable llamada *publicKey*, almacenada como *String* para poder compartirla con el resto de usuarios y que cifren con ella los mensajes (véase Figura 3-15). De este modo, cada vez que se utilice la clase participante desde alguna otra clase, se puede acceder a la clave pública del usuario en cuestión. Esta clase se encarga de aportar todos los datos necesarios para realizar la comunicación de los participantes en la misión.

```

10  class Participante {
11
12      lateinit var id: String
13      lateinit var lon: String
14      lateinit var lat: String
15      lateinit var ip: String
16      lateinit var customInfoWindow : CustomInfoWindow
17      lateinit var marcador: Marker
18      lateinit var publicKey: String
19

```

Figura 3-15 Clase *Participante* modificada para el cifrado asimétrico

A continuación, se presentarán los cambios realizados en la clase *MapsActivity*. Esta clase crea un objeto de la clase *KeyGenerator* y después se invoca el método *generateKeyPair* para generar el par de claves pública y privada que realizarán el cifrado asimétrico.

También ha sido modificado el mensaje broadcast que envían los participantes cada determinado tiempo actualizando su posición. En este mensaje, anteriormente se enviaba el id, la posición GPS (latitud y longitud) y la dirección IP del usuario. Ahora, además de enviar esos parámetros, se añade la clave pública con la que hay que cifrar los mensajes enviados a ese dispositivo (ver Figura 3-16).

```

148
149      //PASA A LA VARIABLE "MESSAGE" NOMBRE,LAT,LONG E IP PARA EMITIR UN BROADCAST A TRAVÉS DE LA CLASE DE COMUNICACIONES
150      var mensajeBroadcast = String()
151      var pubKeyString :String? = keys!!.publicKeyString
152      mensajeBroadcast = mId+"."+mLatitud+"."+mLongitud+"."+ip+"##"+pubKeyString+":"
153
154      var mensajeBroadcastEncriptado :String = encrypt(mensajeBroadcast, hashString(claveAcceso, algorithm: "MD5"))
155      var mensajeEnviado :String = "ATC."+mensajeBroadcastEncriptado
156
157      udpComm.sendUDP(mensajeEnviado)
158

```

Figura 3-16 Contenido del mensaje broadcast

La nueva estructura del mensaje sería, por tanto, la mostrada en la figura Figura 3-17:

ID	:	LAT	,	LONG	,	IP	##	CLAVE PÚBLICA	::
----	---	-----	---	------	---	----	----	---------------	----

Figura 3-17 Estructura del mensaje broadcast

Como *publicKeyString* es una cadena de caracteres, se ha decidido delimitar el inicio de la cadena con un doble '#' y un doble ':' para el final de la cadena. Este nuevo campo implica que al recibir un paquete debemos obtener los datos que se ilustran en la Figura 3-18. Para ello, separamos por un lado Id recibido, latitud recibida, longitud recibida y el resto forma parte de una cadena auxiliar. Dentro de esa cadena auxiliar diferenciamos entre la dirección IP y la clave pública.

```

247     val idRecibido :String = recibido.substring(0, recibido.indexOf( string: ";"))
248     val latRecibido :String = recibido.substring(recibido.indexOf( string: ";") + 1, recibido.indexOf( string: ","))
249     val lonRecibido :String = recibido.substring(recibido.indexOf( string: ",") + 1, recibido.lastIndexOf( string: ","))
250     val cadenaAuxiliar :String = recibido.substring( startIndex: recibido.lastIndexOf( string: ",") + 1)
251     val ipRecibida :String = cadenaAuxiliar.substring(0, cadenaAuxiliar.indexOf( string: "#"))
252     val claveRecibida :String = cadenaAuxiliar.substring(cadenaAuxiliar.indexOf( string: "#") + 2, cadenaAuxiliar.indexOf( string: "::"))
253

```

Figura 3-18 Análisis del mensaje broadcast recibido

Por último, en lo que respecta a la clase *MapsActivity*, cuando seleccionamos a algún usuario y le damos al botón de *chat*, la nueva actividad llamada *chat* obtiene la siguiente información necesaria de este participante (como se muestra en la Figura 3-19) mediante el paso de parámetros con *intent*:

```

177
178 //FUNCIÓN QUE SE EJECUTA AL PULSAR EL BOTÓN DE CHAT EN PANTALLA
179 buttonChat.setOnClickListener { it: View!
180
181     var idChat :String = ventana.participanteSeleccionado
182     for (participante :Participante in participantes) {
183         if (participante.id == idChat) {
184             udpComm.sendUDP(encrypt( strToEncrypt: ("+"+miId+""), hashString(claveAcceso, algorithm: "MD5")))
185             var intent = Intent( packageContext: this@MapsActivity, Chat::class.java)
186             intent.putExtra( name: "ID", idChat)
187             intent.putExtra( name: "IP", participante.ip)
188             intent.putExtra( name: "miId", miId)
189             intent.putExtra( name: "clavePublicaRemoto", participante.publicKey)
190             var privKeyString :String? = keys!!.privateKeyString
191             intent.putExtra( name: "miClavePrivada", privKeyString)
192             startActivity(intent)
193         }
194     }
195 }
196

```

Figura 3-19 Información acerca del participante seleccionado

Ahora, además de todas las variables que se pasan mediante *intent* al *chat*, se incluyen la clave pública del participante remoto y la clave privada del emisor, ambas claves enviadas como *String*. Esto es importante ya que son las claves que necesitamos para cifrar los mensajes que enviamos a ese usuario (su clave pública) como para descifrar lo que se reciba de cualquier usuario (clave privada).

La clase *Chat*, clase encargada de la transmisión de los mensajes entre usuarios, también ha sido modificada para el cifrado de la comunicación entre participantes. En la Figura 3-20 se obtienen los parámetros *intent* en esta clase.

```

58
59     this.keys = KeyGenerator()
60     keys!!.importKeys(pubKeyString, privKeyString)
61

```

Figura 3-20 Obtención de los parámetros *intent* mediante *importKeys*

En la función *clickButtonSend*, se envía a otro usuario a través de la red el mensaje escrito. Mediante esta modificación (ver Figura 3-21), el mensaje que sale del dispositivo móvil *chatMessage* es encriptado por la función *encrypt* de la clase *KeyGenerator*, dado que el objeto *keys* tiene cargadas la clave pública del remitente y la clave privada del emisor, dando lugar al *mensajeEncriptado* que es el que se envía a través de la red.

```
81 fun clickButtonSend(view: View) {
82
83     val editText :EditText! = findViewById<EditText>(R.id.editText)
84
85     val message :String = editText.text.toString()
86     val chatMessage ="$myid: $message"
87
88
89     adapter.add(ChatToItem(text = chatMessage))
90     var mensajeEncriptado :String? = this.keys?.encrypt(chatMessage)
91
92
93     if (mensajeEncriptado != null) {
94         comm.sendUDP(mensajeEncriptado)
95     }
96
97     editText.setText("")
98 }
```

Figura 3-21 Función que permite enviar mensajes cifrados a otros usuarios

Por último, mediante la función *receiveChat* (Figura 3-22), el usuario recibe un mensaje cifrado. La función recibe esta cadena encriptada y mediante la función *decrypt*, se descifra de forma asimétrica ya que este objeto tiene importada, como se explicó anteriormente, la clave privada del receptor de este mensaje.

```
120
121 fun receiveChat() {
122
123     val mensajeRecibido = comm.receiveUDP( broadcast: true)
124     this@Chat.runOnUiThread(java.lang.Runnable {
125
126         val descifrado = this.keys?.decrypt(mensajeRecibido)
127         adapter.add(ChatFromItem(text = descifrado))
128     })
129 }
```

Figura 3-22 Función encargada de descryptar el mensaje recibido

En la Figura 3-23 se muestra un ejemplo de cómo se ve por pantalla una conversación por chat. En este caso Alice está hablando con Bob. Los mensajes de Alice aparecen a la derecha de la imagen y de color gris mientras que los de Bob aparecen a la izquierda de color negro. Ambos mensajes e identificadores son legibles, los de Alice porque representan información local del dispositivo y los de Bob porque ya han sido descifrados.

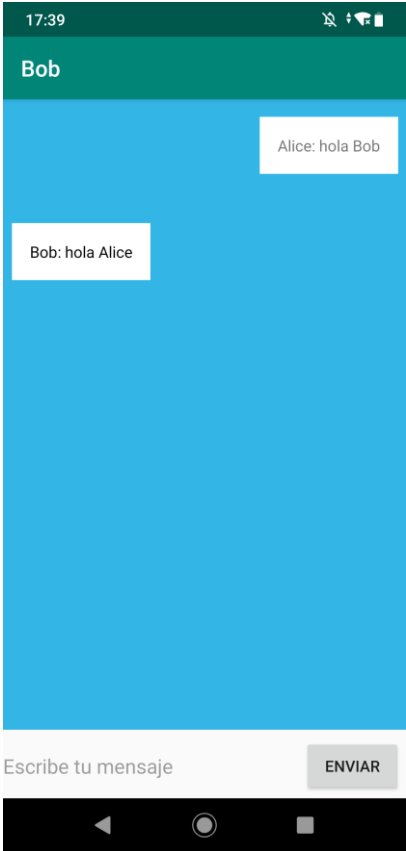


Figura 3-23 Ejemplo de imagen del chat desde el dispositivo de Alice

4 VALIDACIÓN DEL TFG

En este capítulo, se pretende ilustrar la validez de las soluciones implementadas tanto para autenticar usuarios como para ocultar los mensajes entre participantes. Al final de este capítulo se presenta el resultado de una prueba de la aplicación en un escenario lo mas realista posible.

4.1 Validación del sistema de autenticación implementado.

Para la comprobación del correcto funcionamiento del cifrado simétrico haremos uso de una aplicación llamada Wireshark el analizador de protocolo de red más conocido y usado, que nos permitirá interceptar las comunicaciones entre los participantes de la aplicación ATC, capturando los paquetes transmitidos en la red Wi-Fi.

Se validará el cifrado simétrico, esto es, que ningún usuario que desconozca la clave de operación pueda acceder a información del resto de participantes lícitos de ATC mediante:

- La captura de paquetes broadcast antes y después de la implementación de la capa de seguridad realizada en este TFG.
- Demostración práctica en la propia aplicación ATC de acceder a los datos del resto de participantes. Esto tendrá lugar en la prueba de la aplicación.

En primer lugar, se lanza Wireshark desde un ordenador conectado a la misma red local que los participantes de la aplicación ATC. En la Figura 4-1 se muestra la pantalla de inicio de Wireshark.

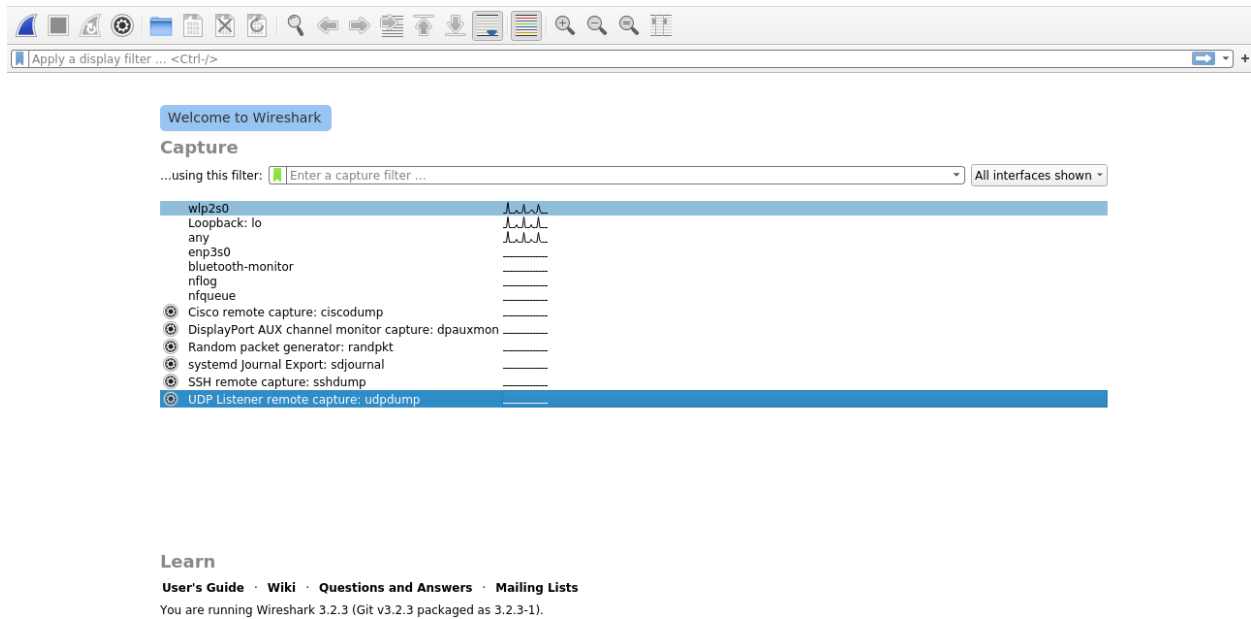


Figura 4-1 Interfaz de inicio de Wireshark

En la página de inicio se debe seleccionar la interfaz de red que queremos monitorizar, esta interfaz de red debe estar conectada a la misma red Wi-Fi que los dispositivos móviles que usan ATC. En este caso, se seleccionó *wlp2s0* que hace referencia a la tarjeta Wi-Fi asociada a la red TFG.

Una vez iniciada la captura de paquetes, como se puede ver en la Figura 4-2, Wireshark comienza a mostrar toda la información y los mensajes que intercambian los dispositivos conectados a la red Wi-Fi seleccionada previamente. En este caso, vamos a ampliar la información del mensaje UDP enviado por la dirección IP de uno de los participantes (192.168.1.169) como mensaje broadcast.

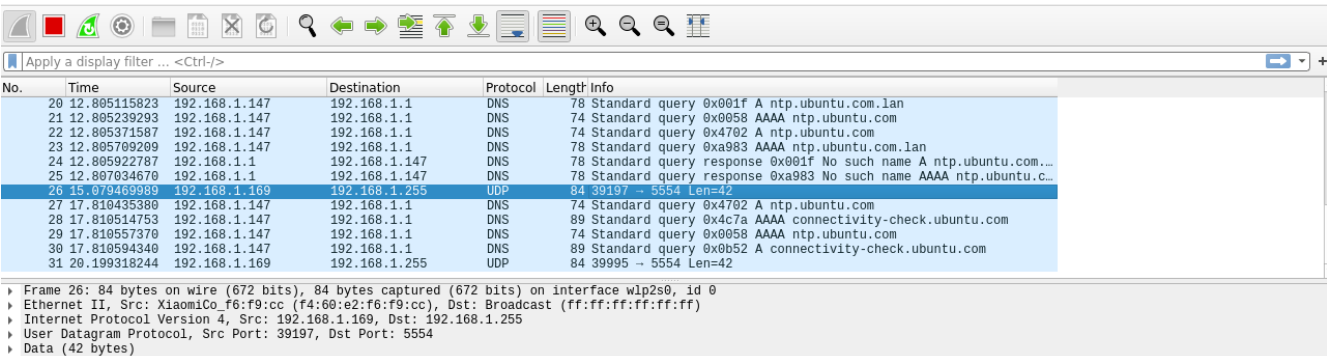


Figura 4-2 Lista de mensajes transmitidos por dispositivos conectados a la red Wi-Fi

La Figura 4-3, nos muestra la información que contiene este paquete UDP, un mensaje broadcast enviado a 192.168.1.169 en claro. El enemigo en estas condiciones podría obtener fácilmente el ID del usuario, en este caso Alice; la latitud y longitud y, por último, la dirección IP del usuario.

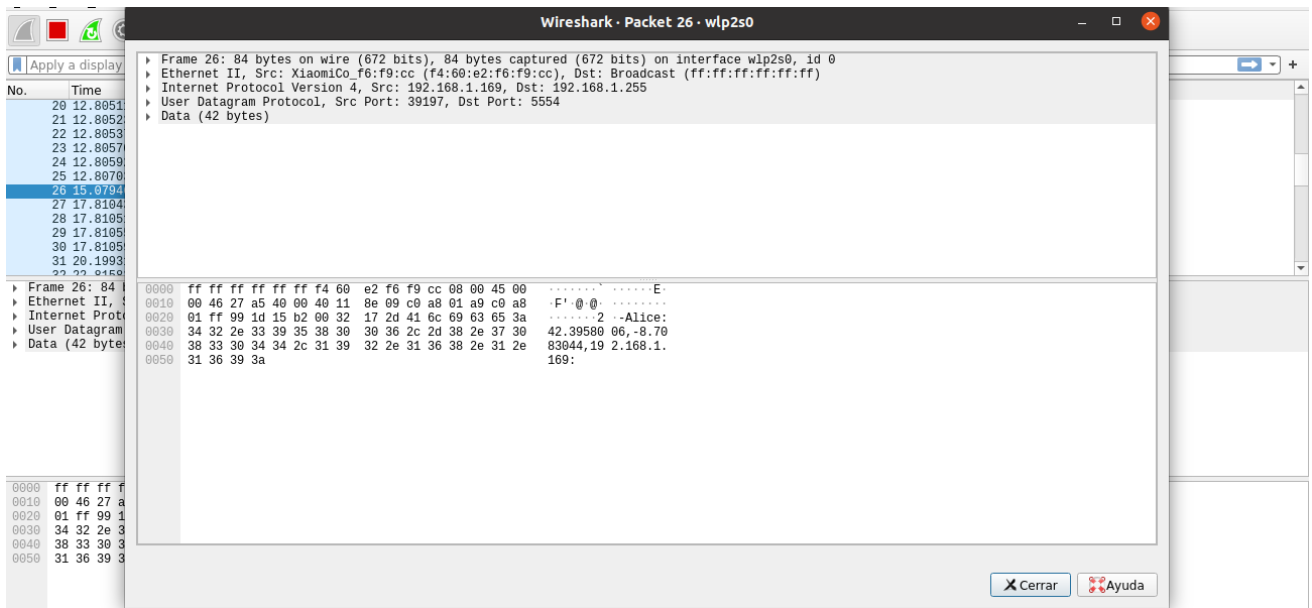


Figura 4-3 Información de un mensaje broadcast enviado por un usuario que utiliza la aplicación sin cifrado simétrico

Todo el proceso de validación del cifrado simétrico presentado hasta ahora ha sido una demostración de cómo, previo a este trabajo, desde cualquier dispositivo es posible acceder a información crítica de los usuarios de ATC, como la posición en la que se encuentran. Si repetimos la operación pero ahora con los dispositivos móviles ejecutando la nueva aplicación ATC (con cifrado simétrico en el intercambio de mensajes broadcast), se distingue la marca inicial del mensaje “ATC.”, pero ahora el resto del mensaje está cifrado (ver Figura 4-4) y, por tanto, que un usuario enemigo no puede hacer uso de dicha información.

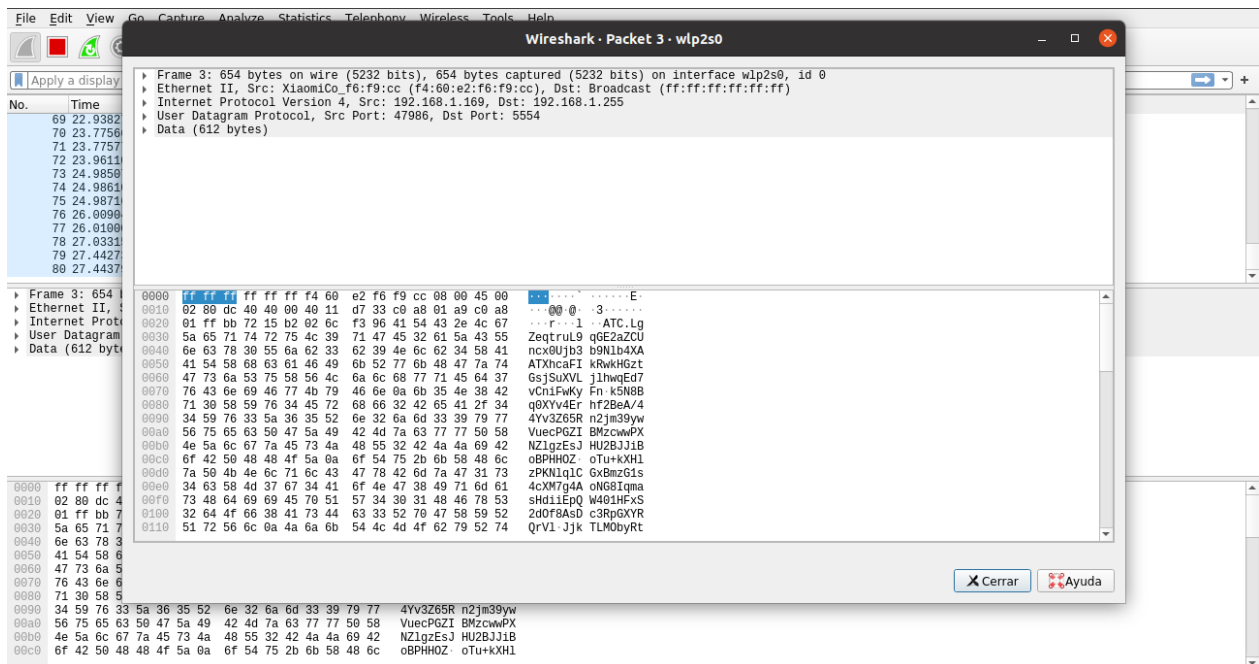


Figura 4-4 Información de un mensaje broadcast enviado por un usuario que utiliza la aplicación con cifrado simétrico

4.2 Validación de la encriptación de mensajes

Para comprobar que el cifrado asimétrico funciona, se ha utilizado también la aplicación Wireshark, pero en este caso apoyada por otra llamada tPacketCapture, esta aplicación se utiliza para poder interceptar los mensajes que se intercambian entre los participantes. Existen otras técnicas para conseguir interceptar mensajes en redes Wi-Fi (por ejemplo, ataques Man-in-the-Middle), pero tPacketCapture nos permite capturar los paquetes UDP que se intercambian los usuarios de ATC de manera sencilla. La aplicación tPacketCapture captura el mensaje enviado y lo almacena. Utilizamos Wireshark para procesar el archivo de paquetes capturados.

Como demostración de que los mensajes están cifrados, se aporta la captura contenida en la Figura 4-5, donde el participante Bob envía a Alice el mensaje “hola”.

```

▶ Frame 169: 391 bytes on wire (3128 bits), 391 bytes captured (3128 bits)
▶ Ethernet II, Src: Google_00:00:00:01 (00:1a:11:00:00:01), Dst: Google_00:00:00:02 (00:1a:11:00:00:02)
▶ Internet Protocol Version 4, Src: 10.8.0.1, Dst: 192.168.43.229
▶ User Datagram Protocol, Src Port: 46626, Dst Port: 6545

0000  00 1a 11 00 00 02 00 1a 11 00 00 01 08 00 45 00  .....E-
0010  01 79 30 4f 00 00 40 11 52 8f 0a 08 00 01 c0 a8  .y00..@. R.....
0020  2b e5 b6 22 19 01 01 65 22 7c 4d 31 52 6b 33 43  +..e "lM1Rk3C
0030  6f 41 42 68 6d 4e 6f 6f 78 58 4c 44 68 45 4c 74  oABhmNoo xXLDhELt
0040  6b 77 4c 4d 32 39 79 4e 57 65 48 73 50 66 74 4f  kWLM29yN WeHsPft0
0050  4e 6f 78 75 76 31 6b 2b 4f 77 70 6d 31 6b 44 4d  Noxuv1k+ Owpm1kDM
0060  65 35 4c 6e 51 72 6a 52 6d 45 56 74 42 41 39 44  e5LnQrJr mEVtBA9D
0070  2f 4d 45 76 65 78 0a 38 66 33 76 73 4f 76 67 38  /MEvex-8 f3vsOvg8
0080  45 4d 7a 74 2f 35 6c 5a 6f 47 70 74 4d 37 52 55  EMzt/5lZ oGptM7RU
0090  68 6a 79 39 36 41 51 64 34 79 42 53 34 66 49 61  hjy96AQd 4yBS4fIa
00a0  67 68 6c 6d 33 36 66 67 48 79 51 58 4c 69 63 68  gh1m36fg HyQXLich
00b0  36 4a 50 68 61 4f 6c 61 77 43 35 73 6c 54 6b 7a  6JPha0la wC5s1Tkz
00c0  6c 75 69 0a 47 35 47 70 53 44 2f 42 31 4a 78 58  lui- G5Gp SD/B1JxX
00d0  5a 48 4c 33 4f 35 62 54 48 6f 7a 6b 62 2b 4e 33  ZHL305bT Hozkb+N3
00e0  59 47 73 30 47 6d 77 5a 38 4c 31 62 44 55 74 38  YGs0GmwZ 8L1bDut8
00f0  6c 5a 37 38 2b 54 61 51 32 63 54 42 75 58 49 4a  lZ78+TaQ 2cTbuXIj
0100  73 35 45 6b 4e 65 34 31 62 51 63 67 2b 4d 65 32  sEkNe41 bQcg+Me2
0110  0a 36 48 2b 37 2b 4d 33 37 2b 70 68 6d 48 55 47  .6H+7+M3 7+phmHUG
0120  41 57 61 46 6c 6f 58 44 54 67 75 58 2b 57 31 6d  AWaFl0XD TguX+W1m
0130  33 77 2b 32 47 2b 49 50 6e 78 56 33 6a 51 6a 4b  3w+2G+IP nxV3jQjK
0140  7a 59 67 6a 37 63 6e 4c 66 58 49 64 4e 61 4c 49  zYgj7cnL fXIdNaLI
0150  42 2f 68 64 36 74 66 39 46 2b 57 74 38 0a 6a 53  B/hd6tf9 F+Wt8-jS
0160  6b 6a 67 37 6e 57 58 32 57 59 50 46 30 4e 63 4d  kJg7nWX2 WYPF0NcM
0170  77 68 66 64 69 37 72 4d 6d 50 79 4e 78 49 49 57  whfd17rM mPyNxIIW
0180  31 39 38 77 3d 3d 0a 198w==

```

Figura 4-5 Mensaje transmitido por ATC encriptado mediante el cifrado asimétrico

Los pasos a seguir hasta obtener esta imagen de la aplicación Wireshark son los mismos que los realizados para obtener las imágenes de la validación del cifrado simétrico.

4.3 Aplicación de ATC en un escenario real

En este último apartado, se va a realizar una demostración del empleo de la aplicación en un escenario lo mas realista posible. En primer lugar, se va a explicar cuales son los recursos materiales de los que se dispone para la prueba. En segundo lugar, se va a comprobar el alcance de la red Wi-Fi realizando un despliegue en las inmediaciones del cuartel Almirante Francisco Moreno (Escuela Naval Militar), es decir, lo separados que pueden llegar a estar los componentes del equipo de Infantería de Marina entre sí. Y en tercer lugar se va a demostrar que alguien que no posee la clave de operación no recibe información de los usuarios legítimos.

4.3.1 Recursos materiales

Los recursos disponibles son un router Wi-Fi LINKSYS WRT 1900 ACS [27] y tres dispositivos móviles Xiaomi Mi A2 Lite [28] cuyas características son detalladas a continuación.

El router LINKSYS que se muestra en la Figura 4-6 posee las siguientes características:

- Procesador de doble núcleo a 1,6 GHz.
- Wireless-AC, hasta 4,3 veces más rápida que la tecnología N.

- Compatibilidad con código abierto.
- 4 antenas de alto rendimiento.
- Doble banda simultánea (2,4 + 5 GHz).
- Unidad de procesamiento central (CPU) de doble núcleo a 1,6 GHz.
- Seguridad inalámbrica avanzada.
- Puerto USB 3.0.
- 4 puertos GIGABIT ETHERNET.
- 512 MB de RAM.



Figura 4-6 Router LINKSYS WRT 1900 ACS [27]

Estas son las características de los tres dispositivos móviles Xiaomi que aparecen en la Figura 4-7.

- Procesador Qualcomm Snapdragon 625 Octa-Core de 2 GHz.
- Procesador de gráficos Adreno 506.
- RAM de 3GB + 32 ROM (Ampliable hasta 256GB).
- Cámara dual de 12 + 5 MP.
- Cámara frontal 5 MP.
- Graba a 1080p/720p a 30fps.
- Batería de 4000 mAh.
- Pantalla de 5,84" FHD (1080x2280).
- Bluetooth 4.2.
- Wi-Fi 2,4G y 5G.
- Wi-Fi Direct y Wi-Fi Display.



Figura 4-7 Xiaomi Mi A2 Lite [28]

4.3.2 Despliegue en la ENM

Para el despliegue se ha decidido separar lo máximo posible los tres dispositivos siempre que todos quedasen conectados a la misma red. El resultado de la prueba ha sido el mostrado en la Figura 4-8.

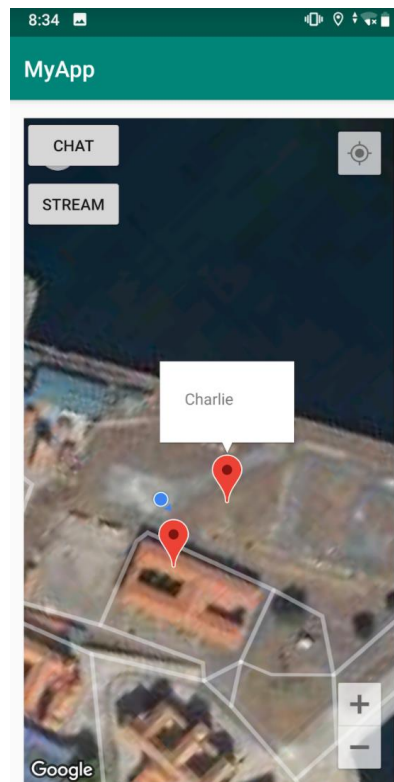


Figura 4-8 Distancia a la que se pueden posicionar los usuarios para estar conectados a la red

En la imagen, aparecen tres usuarios conectados uno de ellos es el portador del móvil que ha capturado la imagen (Bob) que se muestra como un punto azul, el usuario seleccionado (Charlie) y un usuario que se encuentra dentro del edificio (cuartel Almirante Francisco Moreno) junto con el router de la red Wi-Fi (Alice), se ha decidido situar a este último usuario en la posición del router debido a que alguno de los componentes de la misión deberá llevarlo consigo.

Las distancias entre ellos han sido medidas en Google Maps y son las siguientes:

- Entre Alice y Bob hay 44 metros.
- Entre Alice y Charlie hay 56 metros.
- Entre Bob y Charlie hay 50 metros.

Con estas mediciones obtenemos que el alcance máximo de esta red Wi-Fi es de más de 55 metros (teniendo en cuenta la calidad del dispositivo móvil empleado) por lo que la distancia máxima a la que dos usuarios pueden comunicarse supera los 100 metros. Esta afirmación tiene que cumplir que el router se encuentre en el punto medio de esos usuarios. Una de las líneas futuras estará encaminada a aumentar esta distancia máxima a la que se pueda llevar a cabo el flujo de información.

4.3.3 Comprobación de uso de la clave de operación

En esta demostración se ha realizado un intento de acceso a la aplicación por parte de una persona ajena a la misión y el resultado ha sido el mostrado en la Figura 4-9.

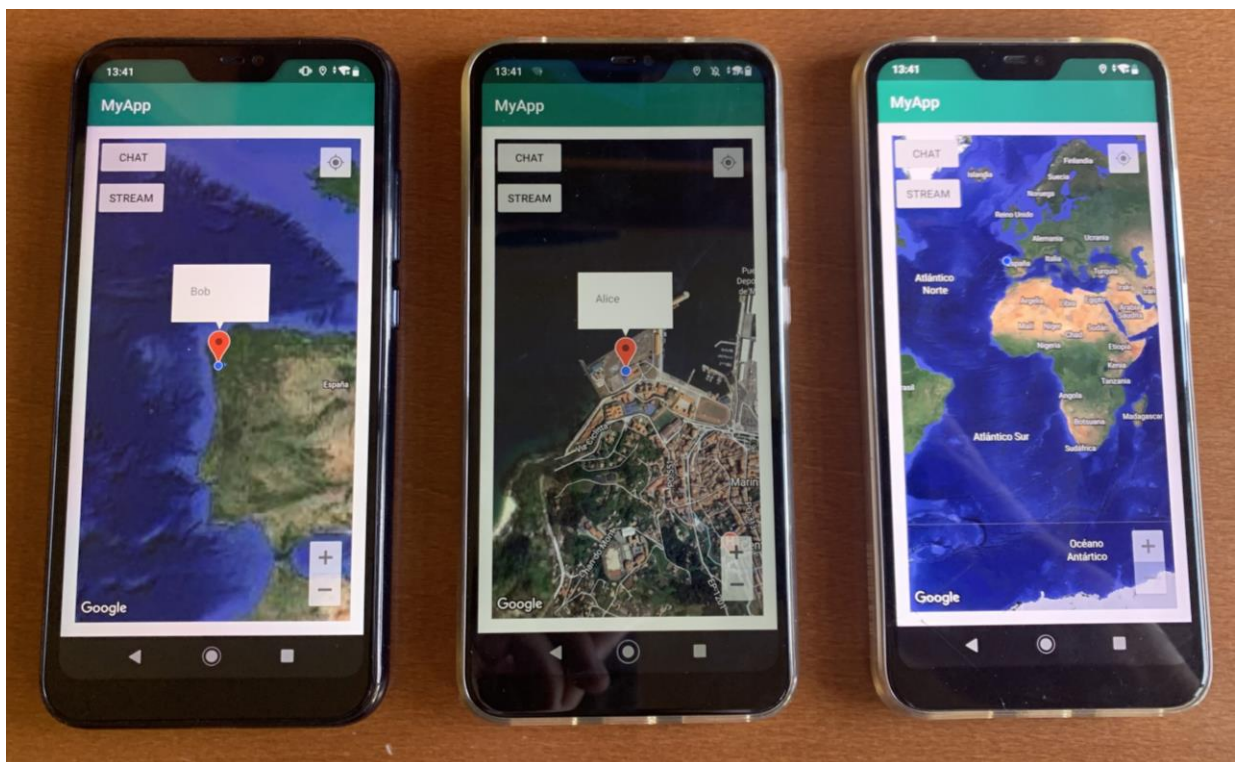


Figura 4-9 Demostración de la seguridad de la aplicación en cuanto al acceso

El dispositivo de la izquierda pertenece a Alice, el del centro pertenece a Bob y el de la derecha pertenece a Charlie. Como se puede apreciar en la imagen, Alice y Bob se pueden ver entre ellos y pueden establecer comunicación entre ellos ya sea mediante mensajería o consumiendo audio y vídeo debido a que forman parte del equipo que realiza la misión y han empleado la contraseña acordada para acceder a la aplicación.

Charlie tiene el papel del enemigo por lo que ha accedido a la aplicación sin conocer la clave de operación de modo que, al entrar a ella, no ve a ningún usuario porque como no ha introducido correctamente la clave de acceso, su dispositivo no tiene la capacidad de descifrar los mensajes broadcast emitidos por el equipo y queda aislado de la transmisión de datos. Queda validada así también la funcionalidad implementada para la autenticación de usuario.

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo, se presentan las conclusiones obtenidas a partir de la realización del trabajo añadiendo además una serie de líneas futuras con la finalidad de sugerir nuevas funcionalidades y mejoras a la aplicación existente en la actualidad.

5.1 Conclusiones

En un principio, ATC era una aplicación con la que un usuario se podía comunicar a través de chat o mediante vídeo con otro participante conectado a la aplicación al mismo tiempo y utilizando la misma red Wi-Fi. Además, se situaba mediante GPS y en el mapa satélite le aparece su posición y la del resto de usuarios que se encuentren conectados en ese mismo instante. Sin embargo, su utilidad se veía mermada ya que toda comunicación establecida era fácilmente interceptable. Esta situación hacía que la aplicación no tuviese la suficiente seguridad en las comunicaciones como para poder ser utilizada en operaciones operaciones o en adiestramiento de personal.

Por esa razón, el objetivo principal de este TFG fue añadir una capa de seguridad a la aplicación ATC que permitiese la comunicación segura en despliegues de Infantería de Marina. Mediante un cifrado simétrico a partir de una clave de operación se consigue impedir el acceso a personal no autorizado (proporciona integridad y confidencialidad). Con un esquema de cifrado asimétrico se cifran las comunicaciones y se impide su interceptación (proporciona integridad, confidencialidad, no repudio y autenticidad), se puede afirmar que el objetivo principal de este Trabajo de Fin de Grado se ha cumplido satisfactoriamente. Este hecho queda respaldado por la validación de ambos cifrados. Destacar además que las soluciones implementadas respetan el diseño de arquitectura distribuida de la aplicación ATC original.

Además del objetivo principal, también se han logrado objetivos personales como son aprender conocimientos de programación en Kotlin, conocer las capacidades de los sistemas Android y poner a prueba los conocimientos adquiridos durante la carrera en asignaturas como Fundamentos de Redes de Ordenadores e Informática para la Ingeniería.

5.2 Líneas futuras

Como posibles líneas futuras que permitan seguir mejorando esta aplicación, se sugieren las siguientes propuestas:

- Mejorar la interfaz de la aplicación haciéndola mas atractiva a ojos del usuario.

- Implementar nuevas funcionalidades a la aplicación que la hagan mas útil y versátil. Dentro de estas posibles nuevas funcionalidades, las expuestas a continuación pueden ser algunas de ellas:
 - Aumentar el radio de cobertura de la aplicación. Este aumento de cobertura de la red se puede hacer mediante una red *mesh* o mallada [29].
Otra forma de aumentar la cobertura es emplear un hardware que posea mayores prestaciones que del que disponemos ahora.
 - Crear notificaciones visibles en el mapa al resto de usuarios, es decir, aportar la capacidad de representar puntos de interés en el mapa satélite para indicar al resto de usuarios cualquier información relevante de esa zona.
 - Implementar unos mensajes predeterminados que ya se emplean en operaciones como son: MEDEVAC [30] mensaje que se envía cuando algún participante de la misión ha sufrido un daño importante y es necesaria evacuación desde una ubicación remota a un hospital, o bien, SPOTREPORT [31] cuando es necesario avisar al mando de que se está accediendo a información relevante para una operación o planeamiento.
 - Desarrollar esta aplicación para dispositivos con otros sistemas operativos como, por ejemplo, iOS.
 - Mejorar la aplicación de *streaming*.
 - Medición del consumo de energía de esta aplicación en el dispositivo móvil debido a que esto puede provocar que la batería se consuma a mayor velocidad suponiendo una reducción del tiempo del dispositivo.

6 BIBLIOGRAFÍA

- [1] L. R. Aguillo, «Diseño y desarrollo de una aplicación en Android para geolocalización y comunicaciones en despliegues de Infantería de Marina,» [En línea]. Available: <http://calderon.cud.uvigo.es/handle/123456789/385>. [Último acceso: 5 Febrero 2021].
- [2] N. Fernández, «Fundamentos de Redes de ordenadores. Tema 8 Introducción a la ciberseguridad,» [En línea]. Available: https://moovi.uvigo.gal/pluginfile.php/297002/mod_resource/content/1/8_Ciberseguridad.pdf. [Último acceso: 19 FEB 2021].
- [3] «Funciones del Distributed Denial of Service,» [En línea]. Available: <https://eldiario.com/2020/07/27/ataques-ddos-un-mecanismo-de-censura-a-medios-de-comunicacion/>. [Último acceso: 7 Marzo 2021].
- [4] «Explicación de Man-in-the-middle,» [En línea]. Available: <https://www.kaspersky.es/blog/que-es-un-ataque-man-in-the-middle/648/>. [Último acceso: 7 Marzo 2021].
- [5] «Información acerca de la aplicación ATAK,» [En línea]. Available: <http://www.ds2.com/solutions/atak>. [Último acceso: 21 Enero 2021].
- [6] «Ampliación de la información sobre la aplicación ATAK,» [En línea]. Available: https://en.wikipedia.org/wiki/Android_Team_Awareness_Kit. [Último acceso: 21 Enero 2021].
- [7] «Imagen de un usuario utilizando ATAK,» [En línea]. Available: https://www.google.com/search?q=atak&sxsrf=ALeKk01PT5X8LyDncJi-V5UwCtYWeXNzgA:1611946066506&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjiosOF58HuAhVbBGMBHVASBbEQ_AUoAnoECAwQBA&biw=718&bih=659#imgsrc=1pAqz61cdUQIRM. [Último acceso: 21 Enero 2021].
- [8] «Uso de ATAK en misión de rescate,» [En línea]. Available: <https://afresearchlab.com/technology/information-technology/tactical-assault-kit-tak/>. [Último acceso: 21 Enero 2021].
- [9] «Información acerca de la aplicación Civ-TAK,» [En línea]. Available: <https://steprimo.com/android/us/app/com.atakmap.app.civ/ATAK-CIV-Android-Team-Awareness-Kit-Civil-Use/>. [Último acceso: 22 Enero 2021].

- [10] «Imagen de la aplicación Civ-TAK en uso,» [En línea]. Available: <https://apkpure.com/es/atak-civ-android-team-awareness-kit-civil-use/com.atakmap.app.civ>. [Último acceso: 21 Enero 2021].
- [11] «Datos acerca del sistema operativo Android con respecto a otros sistemas operativos,» [En línea]. Available: <https://itsoftware.com.co/content/sistemas-operativos-mas-usados/>. [Último acceso: 27 Enero 2021].
- [12] «Imagen del uso de los sistemas operativos en el año 2020,» [En línea]. Available: <https://itsoftware.com.co/content/sistemas-operativos-mas-usados/>. [Último acceso: 27 Enero 2021].
- [13] «Información acerca del lenguaje de programación Kotlin,» [En línea]. Available: <https://ifgeekthen.everis.com/es/que-es-kotlin-y-por-que-deberias-empezar-aprenderlo-ya>. [Último acceso: 27 Enero 2021].
- [14] «Características del lenguaje de programación Kotlin,» [En línea]. Available: <https://developer.android.com/kotlin?hl=es>. [Último acceso: 27 Enero 2021].
- [15] «Definición de IDE,» [En línea]. Available: <https://www.redhat.com/es/topics/middleware/what-is-ide>. [Último acceso: 4 Febrero 2021].
- [16] «Información sobre Android Studio,» [En línea]. Available: <https://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/>. [Último acceso: 4 Febrero 2021].
- [17] «Información sobre IntelliJ IDEA,» [En línea]. Available: https://www.jetbrains.com/idea/promo/?gclid=CjwKCAiAkJKCBhAyEiwAKQBCktFcYIwETHOkhy0G9NvSiI4bLNNDTmT6pErTYq1BQxLY5TOr_yKywBoC4WgQAvD_BwE. [Último acceso: 18 Febrero 2021].
- [18] «Esquema del funcionamiento del cifrado simétrico,» [En línea]. Available: https://virtual.itca.edu.sv/Mediadores/cms/u63_esquema_general_de_un_sistema_de_cifrado_simetrico.html. [Último acceso: 7 Marzo 2021].
- [19] «Técnica de encriptación One-time pad,» [En línea]. Available: https://en.wikipedia.org/wiki/One-time_pad. [Último acceso: 21 Feb 2021].
- [20] «Cifrado simétrico AES,» [En línea]. Available: <https://hardzone.es/tutoriales/rendimiento/cifrado-aes-256-bits-como-funciona/>. [Último acceso: 21 Feb 2021].
- [21] «Esquema de un cifrado asimétrico,» [En línea]. Available: https://www.ecured.cu/Algoritmo_asim%C3%A9trico. [Último acceso: 7 Marzo 2021].
- [22] «Cifrado RSA,» [En línea]. Available: http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/rsa.html. [Último acceso: 21 Feb 2021].
- [23] «Cifrado DSA,» [En línea]. Available: <http://www.dit.upm.es/~pepe/401/index.html#!3917>. [Último acceso: 21 Feb 2021].
- [24] «Funcionalidad de un Intent,» [En línea]. Available: <https://elbauldelprogramador.com/programacion-android-intents-conceptos/#:~:text=Un%20intent%20sirve%20para%20invocar,proveedores%20de%20conteni do%2C%20c%C3%B3digo%20que>. [Último acceso: 8 Marzo 2021].

- [25] «Opción GCM dentro del método de cifrado AES,» [En línea]. Available: https://www.watchguard.com/help/docs/help-center/es-419/Content/es-419/Fireware/mvpn/general/ipsec_algorithms_protocols_c.html?TocPath=Configurar%20los%20Ajustes%20de%20Red%7CT%C3%BAneles%20VPN%20Manuales%20para%20Sucursales%7CAcerca%20de%20la%20VPNs%20de. [Último acceso: 8 Marzo 2021].
- [26] «Explicación del código para generar clave pública y privada,» [En línea]. Available: <https://www.youtube.com/watch?v=jYZ-e-0Jdfs>. [Último acceso: 17 Febrero 2021].
- [27] «Router Wi-Fi empleado,» [En línea]. Available: <https://www.linksys.com/es/p/P-WRT1900ACS/>. [Último acceso: 9 Marzo 2021].
- [28] «Características del Xiaomi Mi A2 Lite,» [En línea]. Available: <https://elandroidelibre.espanol.com/2018/07/xiaomi-mi-a2-xiaomi-mi-a2-lite.html>. [Último acceso: 9 Marzo 2021].
- [29] «Definición de red mallada,» [En línea]. Available: <https://www.xataka.com/especiales/redes-wifi-mesh-que-son-como-funcionan-y-por-que-pueden-mejorar-tu-red-wifi-en-casa>. [Último acceso: 6 Marzo 2021].
- [30] «Definición de MEDEVAC,» [En línea]. Available: <https://www.unidadvictimas.gov.co/es/que-es-el-medevac-los-brigadistas-de-emergencia-estaran-actualizando-este-formato/39128>. [Último acceso: 8 Marzo 2021].
- [31] «Definición de SPOTREPORT,» [En línea]. Available: <https://www.thefreedictionary.com/spot+report>. [Último acceso: 8 Marzo 2021].

ANEXO I: CÓDIGO DE LA APLICACIÓN

En el anexo queda adjuntado todo el código de la aplicación en la que se encuentran las clases utilizadas y las actividades.

En primer lugar, se adjuntan las actividades.

MainActivity:

```
package com.example.myapplication

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnEnter.setOnClickListener {

            var mild: String = editTextId.text.toString()

            var claveAcceso: String = editTextTextPassword.text.toString()

            val intent = Intent(this, MapsActivity::class.java)
            intent.putExtra("Id", mild)
            intent.putExtra("claveAcceso", claveAcceso)
            startActivity(intent)

        }
    }

    fun btnEnter(view: View) {}
}
```

MapsActivity:

```
package com.example.myapplication

import android.Manifest
import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.net.wifi.WifiManager
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.text.format.Formatter
import android.util.Base64
import android.util.Log
import android.view.SurfaceHolder
```

```

import android.view.View
import android.view.WindowManager
import android.widget.Toast
import androidx.core.app.ActivityCompat
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationCallback
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationResult
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.pedro.rtsp.utils.ConnectCheckerRtsp
import com.pedro.rtspserver.RtspServerCamera1
import kotlinx.android.synthetic.main.activity_maps.*
import java.security.*
import javax.crypto.*
import javax.crypto.spec.SecretKeySpec

class MapsActivity : AppCompatActivity(), OnMapReadyCallback, ConnectCheckerRtsp, View.OnClickListener,
SurfaceHolder.Callback {

    private lateinit var mMap: GoogleMap

    var fusedLocationClient: FusedLocationProviderClient? = null

    val participantes = mutableListOf<Participante>()

    var keys: KeyGenerator ? = null

    var locationRequest: LocationRequest? = null

    var callback: LocationCallback? = null
    var miLatitud = ""
    var miLongitud = ""
    var ip = ""
    var mild: String = ""
    var claveAcceso: String = ""

    lateinit var udpComm: Comunicaciones

    lateinit var ventana: CustomInfoWindow

    lateinit var hiloAlertaChat: Thread
    private var rtspServerCamera1: RtspServerCamera1? = null

    fun String.md5(): String{
        return hashString(this,"MD5")
    }
    fun String.sha256():String{
        return hashString(this,"SHA-256")
    }
    fun hashString(input: String, algorithm: String): String{
        return MessageDigest.getInstance(algorithm).digest(input.toByteArray()).fold("",{str, it -> str + "%02x".format(it)})
    }

    @SuppressWarnings("GetInstance")
    fun encrypt(strToEncrypt: String, password: String): String{
        val cipher = Cipher.getInstance("AES")
        val skey = SecretKeySpec(password.toByteArray(), "AES/GCM/NoPadding")

        cipher.init(Cipher.ENCRYPT_MODE, skey)
        val claveEncriptada = cipher.doFinal(strToEncrypt.toByteArray())

        return Base64.encodeToString(claveEncriptada, Base64.DEFAULT)
    }
    @SuppressWarnings("GetInstance")
    fun decrypt(strToDecrypt: String, password: String): String {

```

```

val cipher = Cipher.getInstance("AES")
val skey = SecretKeySpec(password.toByteArray(), "AES/GCM/NoPadding")

cipher.init(Cipher.DECRYPT_MODE, skey)
try {

    val claveDesencriptada = cipher.doFinal(Base64.decode(strToDecrypt, Base64.DEFAULT))
    return String(claveDesencriptada)

} catch (e: Exception) {
}
return ""
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
    setContentView(R.layout.activity_maps)

    this.keys = KeyGenerator()
    this.keys!!.generateKeyPair()

    rtspServerCamera1 = RtspServerCamera1(surfaceView, this, 1935)
    surfaceView.holder.addCallback(this)

    val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as SupportMapFragment

    mapFragment.getMapAsync(this)

    fusedLocationClient = FusedLocationProviderClient(this)

    inicializarLocationRequest()

    udpComm = Comunicaciones()

    var wifiManager: WifiManager

    wifiManager = this.applicationContext?.getSystemService(Context.WIFI_SERVICE) as WifiManager

    var iiplocal = wifiManager.connectionInfo.ipAddress

    ip = Formatter.formatIpAddress(iiplocal)

    udpComm.ipLocal = ip //ASIGNA A LA VAR IPLOCAL EN LA CLASE DE COMUNICACIONES EL VALOR DE LA IP
    DEL TELÉFONO

    udpComm.RemoteHost = ip.substring(0, ip.lastIndexOf(".")) + ".255" //ASIGNA A LAS VAR REMOTEHOST EN LA
    CLASE COMUNICACIONES EL VALOR DE LA IP DEL AP

    callback = object : LocationCallback() {

        @SuppressWarnings("MissingPermission")
        override fun onLocationResult(locationResult: LocationResult?) {
            super.onLocationResult(locationResult)

            mMap.isMyLocationEnabled = true
            mMap.uiSettings.isMyLocationButtonEnabled = true
            mMap.uiSettings.isZoomControlsEnabled = true

            for (ubicacion in locationResult?.locations!!) { //OBTIENE MI UBICACIÓN ACTUAL

                miLatitud = ubicacion.latitude.toString()
                miLongitud = ubicacion.longitude.toString()
            }

            //PASA A LA VARIABLE "MESSAGE" NOMBRE,LAT, LONG E IP PARA EMITIR UN BROADCAST A TRAVÉS DE
            LA CLASE DE COMUNICACIONES

```

```

        var mensajeBroadcast = String()
        var pubKeyString = keys!!.publicKeyString
        mensajeBroadcast = mild+"."+miLatitud+"."+miLongitud+"."+ip+"###"+pubKeyString+"::"

        var mensajeBroadcastEncriptado= encrypt(mensajeBroadcast, hashString(claveAcceso,"MD5"))
        var mensajeEnviado = "ATC."+mensajeBroadcastEncriptado

        udpComm.sendUDP(mensajeEnviado)
    }
}

//INTENT QUE OBTIENE DEL MAINACTIVITY LA VAR "ID" Y LA CLAVE DEL USUARIO QUE INICIA SESIÓN
val objetoIntent: Intent = intent

mild = objetoIntent.getStringExtra("Id")
claveAcceso = objetoIntent.getStringExtra("claveAcceso")

//CIFRADO DE LA CLAVEACCESO
val code = encrypt(claveAcceso, hashString(mild,"MD5"))

//EJECUTA LA ESCUCHA DE MENSAJES BROADCAST Y DE CHAT
hiloAlertaChat = Thread {
    receiveBroadcast()
}
hiloAlertaChat.start()

//FUNCIÓN QUE SE EJECUTA AL PULSAR EL BOTÓN DE CHAT EN PANTALLA
buttonChat.setOnClickListener {

    var idChat = ventana.participanteSeleccionado
    for (participante in participantes) {
        if (participante.id == idChat) {
            udpComm.sendUDP(encrypt("(" + mild + ")", hashString(claveAcceso,"MD5")))
            var intent = Intent(this@MapsActivity, Chat::class.java)
            intent.putExtra("ID", idChat)
            intent.putExtra("IP", participante.ip)
            intent.putExtra("mild", mild)
            intent.putExtra("clavePublicaRemoto", participante.publicKey)
            var privKeyString = keys!!.privateKeyString
            intent.putExtra("miClavePrivada", privKeyString)
            startActivity(intent)
        }
    }
}

//FUNCIÓN QUE SE EJECUTA AL PULSAR EL BOTÓN DE STREAM EN PANTALLA
streaming.setOnClickListener {

    var idStreaming = ventana.participanteSeleccionado
    for (participante in participantes) {
        if (participante.id == idStreaming) {
            val uri: Uri = Uri.parse("rtsp://" + participante.ip + ":1935")
            var intent = Intent(Intent.ACTION_VIEW, uri)
            intent.setPackage("org.videolan.vlc")
            startActivity(intent)
        }
    }
}

fun receiveBroadcast() {

    while (true) {
        // Recibimos el mensaje de broadcast pero también los mensajes de CHAT
        var mensajeRecibidoEncriptado = udpComm.receiveUDP(true)
        mensajeRecibidoEncriptado = mensajeRecibidoEncriptado.substring(0,612)
    }
}

```

```

if (mensajeRecibidoEncriptado.substring(0,4) == "ATC:") {
    println(mensajeRecibidoEncriptado.length)
    //Es nuestro mensaje
    var recibido = decrypt(mensajeRecibidoEncriptado.substring(4, mensajeRecibidoEncriptado.length),
hashString(claveAcceso, "MD5"))

    // Si es un mensaje de CHAT no tiene en la cadena los dos puntos, empieza por "("
    if (recibido.indexOf("(") != -1) {
        this@MapsActivity.runOnUiThread(java.lang.Runnable {

            Toast.makeText(
                this@MapsActivity,
                recibido + "quiere iniciar chat",
                Toast.LENGTH_LONG
            ).show()
        })
        return
    }
    // Si estamos en este punto NO es un mensaje de CHAT
    //PARSEO DEL MENSAJE "RECIBIDO"
    if (recibido.indexOf("::") == -1) {
        return
    }
    val idRecibido = recibido.substring(0, recibido.indexOf(":"))
    val latRecibido = recibido.substring(recibido.indexOf(":") + 1, recibido.indexOf(","))
    val lonRecibido = recibido.substring(recibido.indexOf(",") + 1, recibido.lastIndexOf(","))
    val cadenaAuxiliar = recibido.substring(recibido.lastIndexOf(",") + 1)
    val ipRecibida = cadenaAuxiliar.substring(0, cadenaAuxiliar.indexOf("#"))
    val claveRecibida = cadenaAuxiliar.substring(cadenaAuxiliar.indexOf("#") + 2, cadenaAuxiliar.indexOf("::"))

    this@MapsActivity.runOnUiThread(java.lang.Runnable {
    })
    // Buscamos al usuario, en principio no está en la lista
    var encontrado = 0
    for (participante in participantes) {

        if (participante.id == idRecibido) {

            this@MapsActivity.runOnUiThread(java.lang.Runnable {
                participante.update(latRecibido, lonRecibido, claveRecibida)
                participante.show(mMap, this)
            })
            encontrado = 1
        }
    }
    // Comprobamos que el participante no existe previamente
    if (encontrado == 0) {
        var participante = Participante()
        participante.id = idRecibido
        participante.lat = latRecibido
        participante.lon = lonRecibido
        participante.ip = ipRecibida
        participante.publicKey = claveRecibida

        this@MapsActivity.runOnUiThread(java.lang.Runnable {

            participante.show(mMap, this)
        })
        participantes.add(participante)
    }
}
}
}

private fun inicializarLocationRequest() { //INICIA LA PETICIÓN DE UBICACIÓN CADA 10 SEGUNDOS
    locationRequest = LocationRequest()

```

```

locationRequest?.interval = 1000
locationRequest?.fastestInterval = 500
locationRequest?.priority = LocationRequest.PRIORITY_HIGH_ACCURACY
}

override fun onMapReady(googleMap: GoogleMap) { //CREA UN MAPA TIPO HÍBRIDO (SATÉLITE+CALLES)
    mMap = googleMap
    mMap.mapType = GoogleMap.MAP_TYPE_HYBRID

    obtenerUbicacion()

    ventana = CustomInfoWindow(this)

    mMap.setInfoWindowAdapter(ventana)
}

private fun obtenerUbicacion() {

    if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
    fusedLocationClient?.requestLocationUpdates(locationRequest, callback, null)
}

private fun detenerActualizacionUbicacion() {
    fusedLocationClient?.removeLocationUpdates(callback)
}

override fun onStart() {
    super.onStart()
    obtenerUbicacion()
}

override fun onPause() {
    super.onPause()
    detenerActualizacionUbicacion()
}

override fun surfaceCreated(surfaceHolder: SurfaceHolder) {
    try{
        if (!rtspServerCamera1!!.isStreaming) {
            if (rtspServerCamera1!!.isRecording || rtspServerCamera1!!.prepareAudio() &&
rtspServerCamera1!!.prepareVideo()) {
                rtspServerCamera1!!.startStream()
            } else {
                Toast.makeText(this, "Error preparing stream, This device cant do it", Toast.LENGTH_SHORT)
                    .show()
            }
        }
    } catch (e: Exception) {
        Toast.makeText(this, e.message, Toast.LENGTH_SHORT).show()
    }
}

override fun onNewBitrateRtsp(bitrate: Long) {

}

override fun onConnectionSuccessRtsp() {
    runOnUiThread {
        Toast.makeText(this@MapsActivity, "Connection success", Toast.LENGTH_SHORT).show()
    }
}

```

```
    }  
}  
  
override fun onConnectionFailedRtsp(reason: String) {  
    runOnUiThread {  
        Toast.makeText(this@MapsActivity, "Connection failed. $reason", Toast.LENGTH_SHORT)  
            .show()  
        rtspServerCamera1!!.stopStream()  
    }  
}  
  
override fun onDisconnectRtsp() {  
    runOnUiThread {  
        Toast.makeText(this@MapsActivity, "Disconnected", Toast.LENGTH_SHORT).show()  
    }  
}  
  
override fun onAuthErrorRtsp() {  
    runOnUiThread {  
        Toast.makeText(this@MapsActivity, "Auth error", Toast.LENGTH_SHORT).show()  
        rtspServerCamera1?.stopStream()  
    }  
}  
  
override fun onAuthSuccessRtsp() {  
    runOnUiThread {  
        Toast.makeText(this@MapsActivity, "Auth success", Toast.LENGTH_SHORT).show()  
    }  
}  
override fun onClick(v: View?) {  
  
}  
  
override fun surfaceChanged(holder: SurfaceHolder?, format: Int, width: Int, height: Int) {  
    runOnUiThread {  
        Toast.makeText(this@MapsActivity, "Surface changed", Toast.LENGTH_SHORT).show()  
    }  
}  
  
override fun surfaceDestroyed(holder: SurfaceHolder?) {  
  
}  
}
```

Chat:

```
package com.example.myapplication  
  
import android.content.Intent  
import android.os.Bundle  
import android.view.View  
import android.widget.EditText  
import androidx.appcompat.app.AppCompatActivity  
import com.xwray.groupie.GroupAdapter  
import com.xwray.groupie.GroupViewHolder  
import com.xwray.groupie.Item  
import kotlinx.android.synthetic.main.activity_chat.*  
import kotlinx.android.synthetic.main.chat_from_row.view.*  
import kotlinx.android.synthetic.main.chat_to_row.view.*  
  
class ChatFromItem(val text: String?): Item<GroupViewHolder> {  
  
    override fun bind(viewHolder: GroupViewHolder, position: Int) {  
        viewHolder.itemView.textview_from_row.text = text  
    }  
}
```

```

    }
    override fun getLayout(): Int {
        return R.layout.chat_from_row
    }
}

class ChatToltem(val text: String): Item<GroupieViewHolder>(){

    override fun bind(viewHolder: GroupieViewHolder, position: Int) {
        viewHolder.itemView.textview_to_row.text = text
    }
    override fun getLayout(): Int {
        return R.layout.chat_to_row
    }
}

class Chat : AppCompatActivity() {
    lateinit var comm: Comunicaciones
    lateinit var ip :String
    lateinit var id:String
    lateinit var myid:String
    var pubKeyString: String = ""
    var privKeyString: String = ""

    var adapter = GroupAdapter<GroupieViewHolder>()
    var keys: KeyGenerator ? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chat)

        val objetoIntent: Intent =intent

        ip = objetoIntent.getStringExtra("IP")
        id = objetoIntent.getStringExtra("ID")
        myid = objetoIntent.getStringExtra("mild")
        pubKeyString = objetoIntent.getStringExtra("clavePublicaRemoto")
        privKeyString = objetoIntent.getStringExtra("miClavePrivada")

        this.keys = KeyGenerator()
        keys!!.importKeys(pubKeyString, privKeyString)

        comm= Comunicaciones()
        comm.RemoteHost=ip

        comm.sendUDP("(" +ip+"")

        recycler_view.adapter = adapter

        comm.RemotePort=6545
        supportActionBar?.title = id

        if (comm.checkSocket()) {
            Thread {
                while (true) {
                    receiveChat()
                }
            }.start()
        }
    }

    fun clickButtonSend(view: View) {

        val editText = findViewById<EditText>(R.id.editText)

        val message = editText.text.toString()
        val chatMessage ="$myid: $message"
    }
}

```

```
adapter.add(ChatToItem(text = chatMessage))
var mensajeEncriptado = this.keys?.encrypt(chatMessage)

if (mensajeEncriptado != null) {
    comm.sendUDP(mensajeEncriptado)
}

editText.setText("")
}

fun receiveChat() {

    val mensajeRecibido = comm.receiveUDP(true)
    this@Chat.runOnUiThread(java.lang.Runnable {

        val descifrado = this.keys?.decrypt(mensajeRecibido)
        adapter.add(ChatFromItem(text = descifrado))

    })
}
}
```

Y a continuación, se adjuntarán las clases:

Participante:

```
package com.example.myapplication

import android.content.Context
import android.content.Intent
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.Marker
import com.google.android.gms.maps.model.MarkerOptions

class Participante {

    lateinit var id: String
    lateinit var lon: String
    lateinit var lat: String
    lateinit var ip: String
    lateinit var customInfoWindow : CustomInfoWindow
    lateinit var marcador: Marker
    lateinit var publicKey: String

    open fun show(map: GoogleMap, contexto: Context){

        val miPosicion = LatLng(lat.toDouble(), lon.toDouble())
        marcador=map.addMarker(MarkerOptions().position(miPosicion).title("$id"))

        marcador.showInfoWindow()
    }

    open fun update(latRecibido:String?, lonRecibido:String?, claveRecibida: String?){
        lat= latRecibido!!
        lon= lonRecibido!!
        publicKey= claveRecibida!!
        marcador.remove()
    }
}
```

KeyGenerator:

```
package com.example.myapplication
```

```

import android.util.Base64
import android.util.Log
import java.security.KeyFactory
import java.security.KeyPairGenerator
import java.security.PrivateKey
import java.security.PublicKey
import java.security.interfaces.RSAPublicKey
import java.security.spec.PKCS8EncodedKeySpec
import java.security.spec.X509EncodedKeySpec
import javax.crypto.Cipher

@Suppress("UNREACHABLE_CODE")
class KeyGenerator {

    var publicKey: PublicKey? = null
    var publicKeyString: String? = null
    var privateKey: PrivateKey? = null
    var privateKeyString: String? = null
    var decryptedString: String? = null
    lateinit var encryptedByte: ByteArray
    lateinit var decryptedByte: ByteArray
    var message = ""

    companion object {
        const val CRYPTO_METHOD = "RSA"
        const val CRYPTO_BITS = 2048
        private const val OPCION_RSA = "RSA/ECB/OAEPWithSHA1AndMGF1Padding"
    }

    fun onCreate() {
        try {
            generateKeyPair()
            val mensajeEncriptado = encrypt(message)
            Log.d("TAG1", "mensaje encriptado -> $mensajeEncriptado")
            val mensajeDesencriptado = decrypt(mensajeEncriptado)
            Log.d("TAG1", "mensaje desencriptado -> $mensajeDesencriptado")
        } catch (e: Exception) {
        }
    }

    @Throws(Exception::class)
    public fun generateKeyPair() {
        //GENERA E INICIALIZA LAS CLAVES PÚBLICA Y PRIVADA
        var kpg = KeyPairGenerator.getInstance(CRYPTO_METHOD)
        kpg.initialize(CRYPTO_BITS)
        //GUARDAMOS LAS CLAVES EN kp
        var kp = kpg.generateKeyPair()

        var encoder = java.util.Base64.getEncoder()

        this.privateKeyString = encoder.encodeToString(kp.getPrivate().getEncoded())
        this.publicKeyString = encoder.encodeToString(kp.getPublic().getEncoded())
    }

    @Throws(Exception::class)
    public fun importKeys(pubKey:String?,privKey:String?) {

        val kf: KeyFactory = KeyFactory.getInstance("RSA")
        val keySpecPKCS8: PKCS8EncodedKeySpec =
        PKCS8EncodedKeySpec(java.util.Base64.getDecoder().decode(privKey))
    }

```

```
val keySpecX509: X509EncodedKeySpec = X509EncodedKeySpec(java.util.Base64.getDecoder().decode(pubKey))

publicKey = kf.generatePublic(keySpecX509) as RSAPublicKey
privateKey = kf.generatePrivate(keySpecPKCS8)
}

@Throws(Exception::class)
public fun encrypt(chatMessage: String): String {
    val cipher = Cipher.getInstance(OPCION_RSA)
    cipher.init(Cipher.ENCRYPT_MODE, this.publicKey)
    encryptedByte = cipher.doFinal(chatMessage.toByteArray())
    return Base64.encodeToString(encryptedByte, Base64.DEFAULT)
}

@Throws(Exception::class)
public fun decrypt(mensajeRecibido: String): String {
    val cipher = Cipher.getInstance(OPCION_RSA)
    cipher?.init(Cipher.DECRYPT_MODE, this.privateKey)
    decryptedByte = cipher!!.doFinal(Base64.decode(mensajeRecibido, Base64.DEFAULT))
    decryptedString = String(decryptedByte)
    return decryptedString!!
}
}
```

Comunicaciones:

```
package com.example.myapp

import android.os.StrictMode
import java.io.IOException
import java.net.DatagramPacket
import java.net.DatagramSocket
import java.net.InetAddress
import java.net.SocketException
import java.security.spec.PKCS8EncodedKeySpec

class Comunicaciones {

    var RemoteHost = "" //dir IP telefonos conectados al AP
    var RemotePort: Int = 5554 // numero puerto aleatorio por encima de 1024
    var ipLocal = ""
    var ipRecibido = ""

    open fun sendUDP(messageStr: String) {

        // Hack Prevent crash (sending should be done using an async task)
        val policy = StrictMode.ThreadPolicy.Builder().permitAll().build()
        StrictMode.setThreadPolicy(policy)

        try {
            //Open a port to send the package
            val socket = DatagramSocket()
            socket.broadcast = true
            val sendData = messageStr.toByteArray()
            val sendPacket = DatagramPacket(sendData, sendData.size, InetAddress.getByName(RemoteHost), RemotePort)
            socket.send(sendPacket)

            //println("fun sendBroadcast: packet sent to: " + InetAddress.getByName(RemoteHost) + ":" + RemotePort)
        } catch (e: IOException) {
            println("Error en envio mensaje UDP "+e.message.toString())
        }
    }
}
```

```

open fun checkSocket(): Boolean {
    try {
        var socket = DatagramSocket(RemotePort)
        socket.close()
        return true
    } catch (e: SocketException) {
        return false
    }
    return false
}
open fun receiveUDP(broadcast:Boolean):String {

    val charset = Charsets.UTF_8
    val buffer = ByteArray(2048)
    var socket: DatagramSocket? = null

    try {

        //Mantiene un socket abierto escuchando el tráfico UDP destinado a este puerto
        if (broadcast) {
            socket = DatagramSocket(RemotePort)
            socket.broadcast = true
        } else {
            socket = DatagramSocket(RemotePort)//, InetAddress.getByName(RemoteHost))
        }
        val packet = DatagramPacket(buffer, buffer.size)
        do {
            socket.receive(packet)
            //println(packet.data.toString(charset))
        } while (packet.address.toString().substring(1) == ipLocal)

        ipRecibido = packet.address.toString().substring(1);
        return packet.data.toString(charset)

    } catch (e: Exception) {
        //e.printStackTrace()
        return e.printStackTrace().toString()

    } finally {
        socket?.close() //PUERTO CERRADO
    }
    return ""
}
}

```

CustomInfoWindow:

```

package com.example.myapp

import android.app.Activity
import android.content.Context
import android.view.View
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.Marker
import kotlinx.android.synthetic.main.custom_window.view.*

class CustomInfoWindow(val context: Context) : GoogleMap.InfoWindowAdapter {

    lateinit var participanteSeleccionado: String

    override fun getInfoContents(p0: Marker?): View {

        var mInfoView = (context as Activity).layoutInflater.inflate(R.layout.custom_window, null)
    }
}

```

```
mInfoView.editTextFormInfoWindow.text = p0!!.title
participanteSeleccionado = p0.title

return mInfoView

}

override fun getInfoWindow(p0: Marker?): View? {
    return null
}
}
```

RTSPserver:

```
package com.pedro.rtspserver

import android.content.Context
import android.media.MediaCodec
import android.net.wifi.WifiManager
import android.util.Log
import com.pedro.rtsp.rtsp.Protocol
import com.pedro.rtsp.rtsp.RtspSender
import com.pedro.rtsp.utils.ConnectCheckerRtsp
import java.io.*
import java.net.InetAddress
import java.net.ServerSocket
import java.net.Socket
import java.net.SocketException
import java.nio.ByteBuffer

class RtspServer(context: Context, private val connectCheckerRtsp: ConnectCheckerRtsp,
    val port: Int) {

    private val TAG = "RtspServer"
    private lateinit var server: ServerSocket
    val serverIp = getServerIp(context)
    var sps: ByteArray? = null
    var pps: ByteArray? = null
    var vps: ByteArray? = null
    var sampleRate = 32000
    var isStereo = true
    private val clients = mutableListOf<Client>()
    private var thread: Thread? = null

    fun startServer() {
        thread = Thread {
            server = ServerSocket(port)
            while (!Thread.interrupted()) {
                Log.i(TAG, "Server started $serverIp:$port")
                try {
                    val client =
                        Client(
                            server.accept(), serverIp, port, connectCheckerRtsp, sps, pps, vps, sampleRate,
                            isStereo
                        )
                    client.start()
                    clients.add(client)
                } catch (e: SocketException) {
                    Log.e(TAG, "Error", e)
                    break
                } catch (e: IOException) {
                    Log.e(TAG, e.message ?: "")
                    continue
                }
            }
        }
        Log.i(TAG, "Server finished")
    }
}
```

```

    thread?.start()
}

fun stopServer() {
    clients.forEach { it.stopClient() }
    clients.clear()
    thread?.interrupt()
    try {
        thread?.join(100)
    } catch (e: InterruptedException) {
        thread?.interrupt()
    }
    thread = null
    if (!server.isClosed) server.close()
}

fun sendVideo(h264Buffer: ByteBuffer, info: MediaCodec.BufferInfo) {
    clients.forEach {
        if (it.isAlive && it.canSend) {
            it.rtspSender.sendVideoFrame(h264Buffer.duplicate(), info)
        }
    }
}

fun sendAudio(aacBuffer: ByteBuffer, info: MediaCodec.BufferInfo) {
    clients.forEach {
        if (it.isAlive && it.canSend) {
            it.rtspSender.sendAudioFrame(aacBuffer.duplicate(), info)
        }
    }
}

fun setVideoInfo(sps: ByteBuffer, pps: ByteBuffer, vps: ByteBuffer?) {
    this.sps = getData(sps)
    this.pps = getData(pps)
    this.vps = getData(vps) //H264 has no vps so if not null assume H265
}

private fun getData(byteBuffer: ByteBuffer?): ByteArray? {
    return if (byteBuffer != null) {
        val bytes = ByteArray(byteBuffer.capacity() - 4)
        byteBuffer.position(4)
        byteBuffer.get(bytes, 0, bytes.size)
        bytes
    } else {
        null
    }
}

private fun getServerIp(context: Context): String {
    val wm = context.applicationContext.getSystemService(Context.WIFI_SERVICE) as WifiManager
    return InetAddress.getByAddress(ByteArray(4) { i ->
        wm.connectionInfo.ipAddress.shr(i * 8).and(255).toByte()
    }).hostAddress
}

internal class Client(private val socket: Socket, serverIp: String, serverPort: Int,
    connectCheckerRtsp: ConnectCheckerRtsp, private val sps: ByteArray?,
    private val pps: ByteArray?, private val vps: ByteArray?, private val sampleRate: Int,
    isStereo: Boolean) : Thread() {

    private val TAG = "Client"
    private var cSeq = 0
    private val output = BufferedWriter(OutputStreamWriter(socket.getOutputStream()))
    private val input = BufferedReader(InputStreamReader(socket.getInputStream()))
    val rtspSender = RtspSender(connectCheckerRtsp)
    private val protocol: Protocol = Protocol.UDP
    private val commandsManager =
        ServerCommandManager(

```

```

        serverIp,
        serverPort,
        socket.inetAddress.hostAddress
    )
    var canSend = false

    init {
        commandsManager.setIsStereo(isStereo)
    }

    override fun run() {
        super.run()
        Log.i(TAG, "New client ${commandsManager.clientIp}")
        while (!interrupted()) {
            try {
                val request = commandsManager.getRequest(input)
                cSeq = commandsManager.getCSeq(request) //update cSeq
                if (cSeq == -1) { //If cSeq parsed fail send error to client
                    output.write(commandsManager.createError(500, cSeq))
                    output.flush()
                    continue
                }
                val action = request.split("\n")[0]
                Log.i(TAG, request)
                val response = commandsManager.createResponse(action, request, cSeq)
                Log.i(TAG, response)
                output.write(response)
                output.flush()

                if (action.contains("play", true)) {
                    rtspSender.setSocketsInfo(protocol, commandsManager.videoClientPorts,
                        commandsManager.audioClientPorts)
                    rtspSender.setVideoInfo(sps, pps, vps)
                    rtspSender.setAudioInfo(sampleRate)
                    rtspSender.setDataStream(socket.getOutputStream(), commandsManager.clientIp)
                    rtspSender.setVideoPorts(commandsManager.videoPorts[0], commandsManager.videoPorts[1])
                    rtspSender.setAudioPorts(commandsManager.audioPorts[0], commandsManager.audioPorts[1])
                    rtspSender.start()
                    canSend = true
                }
            } catch (e: SocketException) { // Client has left
                Log.e(TAG, "Client disconnected")
                break
            } catch (e: Exception) {
                Log.e(TAG, "Unexpected error")
            }
        }
    }

    fun stopClient() {
        canSend = false
        rtspSender.stop()
        interrupt()
        try {
            join(100)
        } catch (e: InterruptedException) {
            interrupt()
        } finally {
            socket.close()
        }
    }
}

```

RTSPserverCamera1:

```

package com.pedro.rtspserver

import android.content.Context
import android.media.MediaCodec
import android.os.Build
import android.view.SurfaceView
import android.view.TextureView
import androidx.annotation.RequiresApi
import com.pedro.encoder.utils.CodecUtil
import com.pedro.rtplibrary.base.Camera1Base
import com.pedro.rtplibrary.view.LightOpenGLView
import com.pedro.rtplibrary.view.OpenGLView
import com.pedro.rtsp.rtsp.VideoCodec
import com.pedro.rtsp.utils.ConnectCheckerRtsp
import java.nio.ByteBuffer

class RtspServerCamera1 : Camera1Base {

    private val rtspServer: RtspServer

    constructor(surfaceView: SurfaceView, connectCheckerRtsp: ConnectCheckerRtsp, port: Int) : super(
        surfaceView) {
        rtspServer = RtspServer(surfaceView.context, connectCheckerRtsp, port)
    }

    constructor(textureView: TextureView, connectCheckerRtsp: ConnectCheckerRtsp, port: Int) : super(
        textureView) {
        rtspServer = RtspServer(textureView.context, connectCheckerRtsp, port)
    }

    @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN_MR2)
    constructor(openGLView: OpenGLView, connectCheckerRtsp: ConnectCheckerRtsp, port: Int) : super(
        openGLView) {
        rtspServer = RtspServer(openGLView.context, connectCheckerRtsp, port)
    }

    @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN_MR2)
    constructor(lightOpenGLView: LightOpenGLView, connectCheckerRtsp: ConnectCheckerRtsp,
        port: Int) : super(lightOpenGLView) {
        rtspServer = RtspServer(lightOpenGLView.context, connectCheckerRtsp, port)
    }

    @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN_MR2)
    constructor(context: Context, connectCheckerRtsp: ConnectCheckerRtsp, port: Int) : super(
        context) {
        rtspServer = RtspServer(context, connectCheckerRtsp, port)
    }

    fun setVideoCodec(videoCodec: VideoCodec) {
        videoEncoder.type =
            if (videoCodec == VideoCodec.H265) CodecUtil.H265_MIME else CodecUtil.H264_MIME
    }

    fun getEndPointConnection(): String = "rtsp://${rtspServer.serverIp}:${rtspServer.port}"

    override fun setAuthorization(user: String, password: String) { //not developed
    }

    fun startStream() {
        super.startStream("")
    }

    override fun prepareAudioRtp(isStereo: Boolean, sampleRate: Int) {
        rtspServer.isStereo = isStereo
        rtspServer.sampleRate = sampleRate
    }

    override fun startStreamRtp(url: String) { //unused

```

```
}

override fun stopStreamRtp() {
    rtspServer.stopServer()
}

override fun getAacDataRtp(aacBuffer: ByteBuffer, info: MediaCodec.BufferInfo) {
    rtspServer.sendAudio(aacBuffer, info)
}

override fun onSpsPpsVpsRtp(sps: ByteBuffer, pps: ByteBuffer, vps: ByteBuffer?) {
    val newSps = sps.duplicate()
    val newPps = pps.duplicate()
    val newVps = vps?.duplicate()
    rtspServer.setVideoInfo(newSps, newPps, newVps)
    rtspServer.startServer()
}

override fun getH264DataRtp(h264Buffer: ByteBuffer, info: MediaCodec.BufferInfo) {
    rtspServer.sendVideo(h264Buffer, info)
}

/**
 * Unused functions
 */
@Throws(RuntimeException::class)
override fun resizeCache(newSize: Int) {
}

override fun shouldRetry(reason: String?): Boolean = false

override fun reConnect(delay: Long) {
}

override fun setReTries(reTries: Int) {
}

override fun getCacheSize(): Int = 0

override fun getSentAudioFrames(): Long = 0

override fun getSentVideoFrames(): Long = 0

override fun getDroppedAudioFrames(): Long = 0

override fun getDroppedVideoFrames(): Long = 0

override fun resetSentAudioFrames() {
}

override fun resetSentVideoFrames() {
}

override fun resetDroppedAudioFrames() {
}

override fun resetDroppedVideoFrames() {
}
}
```

ServerCommandManager:

```
package com.pedro.rtspserver

import android.util.Log
import com.pedro.rtsp.rtsp.Body
import com.pedro.rtsp.rtsp.CommandsManager
```

```

import java.io.BufferedReader
import java.io.IOException
import java.net.SocketException
import java.util.regex.Pattern

class ServerCommandManager(private val serverIp: String, private val serverPort: Int,
    val clientIp: String?) : CommandsManager() {

    private val TAG = "ServerCommandManager"
    var audioPorts = ArrayList<Int>()
    var videoPorts = ArrayList<Int>()

    fun createResponse(action: String, request: String, cSeq: Int): String {
        return when {
            action.contains("options", true) -> createOptions(cSeq)
            action.contains("describe", true) -> createDescribe(cSeq)
            action.contains("setup", true) -> {
                if (loadPorts(request)) createSetup(cSeq) else createError(500, cSeq)
            }
            action.contains("play", true) -> createPlay(cSeq)
            action.contains("pause", true) -> createPause(cSeq)
            action.contains("teardown", true) -> createTeardown(cSeq)
            else -> createError(400, cSeq)
        }
    }

    private fun loadPorts(request: String): Boolean {
        val ports = ArrayList<Int>()
        val portsMatcher =
            Pattern.compile("client_port=(\\d+)(?:-(\\d+))?", Pattern.CASE_INSENSITIVE).matcher(request)
        if (portsMatcher.find()) {
            portsMatcher.group(1)?.toInt()?.let { ports.add(it) }
            portsMatcher.group(2)?.toInt()?.let { ports.add(it) }
        } else {
            Log.e(TAG, "UDP ports not found")
            return false
        }
        val trackMatcher = Pattern.compile("trackID=(\\w+)", Pattern.CASE_INSENSITIVE).matcher(request)
        if (trackMatcher.find()) {
            val track = trackMatcher.group(1)?.toInt()
            if (track == 0) { //audio ports
                audioPorts.clear()
                audioPorts.add(ports[0])
                audioPorts.add(ports[1])
            } else { //video ports
                videoPorts.clear()
                videoPorts.add(ports[0])
                videoPorts.add(ports[1])
            }
        } else {
            Log.e(TAG, "Track id not found")
            return false
        }
        Log.i(TAG, "Video ports: $videoPorts")
        Log.i(TAG, "Audio ports: $audioPorts")
        return true
    }

    fun getCSeq(request: String): Int {
        val cSeqMatcher =
            Pattern.compile("CSeq\\s*:\\s*(\\d+)", Pattern.CASE_INSENSITIVE).matcher(request)
        return if (cSeqMatcher.find()) {
            cSeqMatcher.group(1)?.toInt() ?: -1
        } else {
            Log.e(TAG, "cSeq not found")
            return -1
        }
    }
}

```

```

@Throws(IOException::class, IllegalStateException::class, SocketException::class) fun getRequest(
    input: BufferedReader): String {
    var request = ""
    var line: String? = input.readLine()
    while (line != null && line.length > 3) {
        request += "$line\n"
        line = input.readLine()
    }
    return request
}

private fun createStatus(code: Int): String {
    return when (code) {
        200 -> "200 OK"
        400 -> "400 Bad Request"
        401 -> "401 Unauthorized"
        404 -> "404 Not Found"
        500 -> "500 Internal Server Error"
        else -> "500 Internal Server Error"
    }
}

fun createError(code: Int, cSeq: Int): String {
    return "RTSP/1.0 ${createStatus(
        code)}\r\n" + "Server: pedroSG94 Server\r\n" + "Cseq: $cSeq\r\n\r\n"
}

private fun createHeader(cSeq: Int): String {
    return "RTSP/1.0 ${createStatus(200)}\r\n" + "Server: pedroSG94 Server\r\n" + "Cseq: $cSeq\r\n"
}

private fun createOptions(cSeq: Int): String {
    return createHeader(cSeq) + "Public: DESCRIBE,SETUP,TEARDOWN,PLAY,PAUSE\r\n\r\n"
}

private fun createDescribe(cSeq: Int): String {
    val body = createBody()
    return createHeader(
        cSeq) + "Content-Length: ${body.length}\r\n" + "Content-Base: $serverIp:$serverPort\r\n" + "Content-Type:
application/sdp\r\n\r\n" + body
}

private fun createBody(): String {
    val bodyAudio = Body.createAacBody(trackAudio, sampleRate, isStereo)
    val bodyVideo = Body.createH264Body(trackVideo, "Z0KAHtoHgUZA", "aM4NiA==")
    return "v=0\r\n" + "o=- 0 0 IN IP4 $serverIp\r\n" + "s=Unnamed\r\n" + "i=N/A\r\n" + "c=IN IP4 $clientIp\r\n" + "t=0 0\r\n" +
"a=recvonly\r\n" + bodyAudio + bodyVideo + "\r\n"
}

override fun createSetup(cSeq: Int): String {
    return createHeader(
        cSeq) + "Content-Length: 0\r\n" + "Transport: RTP/AVP/UDP;unicast;destination=$clientIp;client_port=8000-
8001;server_port=39000-35968;ssrc=46a81ad7;mode=play\r\n" + "Session: 1185d20035702ca\r\n" + "Cache-Control: no-
cache\r\n\r\n"
}

private fun createPlay(cSeq: Int): String {
    return createHeader(
        cSeq) + "Content-Length: 0\r\n" + "RTP-Info: url=rtsp://$serverIp:$serverPort\r\n" + "Session: 1185d20035702ca\r\n\r\n"
}

private fun createPause(cSeq: Int): String {
    return createHeader(cSeq) + "Content-Length: 0\r\n\r\n"
}

private fun createTeardown(cSeq: Int): String {
    return createHeader(cSeq) + "\r\n"
}

```

}
}