Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Plan de Trabajo

Desarrollo de un Sistema de Autenticación de Segundo Factor Utilizando D2OTP mediante un Canal Alternativo

Autor: Luis Pérez López

Tutor(a): Miguel Jiménez Gañán

Madrid, Julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Desarrollo de un Sistema de Autenticación de Segundo Factor Utilizando D2OTP mediante un Canal Alternativo
Julio 2025

Autor: Luis Pérez López

Tutor:
Miguel Jiménez Gañán
DLSIIS
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En la actualidad, la seguridad en los sistemas de autenticación cobra una importancia crítica debido al aumento de ciberataques y al uso generalizado de servicios digitales. Este Trabajo de Fin de Grado se centra en el diseño e implementación de un sistema de autenticación de segundo factor (2FA) basado en el protocolo **D2OTP (Doble Dinámica OTP)**, que introduce un enfoque innovador para mejorar la seguridad mediante claves efímeras y autenticación mutua, sin depender de criptografía asimétrica ni infraestructura PKI.

El proyecto abarca el desarrollo completo de una solución 2FA que incluye una **API RESTful** construida con **FastAPI**, una **aplicación móvil autenticadora multiplataforma** desarrollada con **Kotlin Multiplatform Mobile (KMM)**, y una **infraestructura en la nube basada en Firebase** para almacenamiento y mensajería. El sistema permite verificar operaciones sensibles a través de un canal alternativo cifrado, utilizando el protocolo D2OTP, que garantiza la confidencialidad, integridad y autenticidad de los mensajes intercambiados entre cliente y servidor.

Se ha implementado una **librería criptográfica D2OTP** tanto en Python (para el backend) como en Kotlin (para la app móvil), que permite codificar y decodificar los mensajes OTP con claves efímeras. El protocolo D2OTP evita la reutilización de secretos, previene ataques de phishing y man-in-the-middle, y cumple con regulaciones como la directiva PSD2 en cuanto a autenticación reforzada y firma dinámica.

El sistema fue sometido a un análisis exhaustivo de requisitos funcionales y no funcionales, así como a un modelo de amenazas basado en STRIDE y OWASP Mobile Top 10. Se identificaron y mitigaron múltiples riesgos asociados a la suplantación de identidad, divulgación de información, manipulación de datos y elevación de privilegios, estableciendo un marco sólido de protección integral.

En cuanto a la implementación, se construyó una arquitectura modular y escalable con los siguientes componentes: una app cliente que solicita la autenticación, un backend que orquesta el flujo de verificación y una app 2FA que permite al usuario aprobar o rechazar la operación. La comunicación se realiza mediante mensajes cifrados (M1, M2, M3) siguiendo el flujo definido por D2OTP, reforzados por un canal alternativo (push o SMS).

Aunque el sistema es funcional y robusto, se identificaron algunas limitaciones, como la falta de integración completa de notificaciones push y la ausencia de mecanismos de recuperación en caso de pérdida del dispositivo autenticador. Estas carencias no comprometen la seguridad del sistema, pero representan áreas de mejora para futuras versiones.

En conclusión, este proyecto demuestra la viabilidad y ventajas de aplicar el protocolo D2OTP como solución de segundo factor en contextos donde se requiere alta seguridad, pero se desea evitar la complejidad y coste de los

sistemas basados en criptografía asimétrica. La solución desarrollada proporciona una base sólida para futuras investigaciones y despliegues de sistemas de autenticación avanzados, adaptables y centrados en la experiencia del usuario.

Abstract

Nowadays, security in authentication systems is critically important due to the rise in cyberattacks and the widespread use of digital services. This Final Degree Project focuses on the design and implementation of a two-factor authentication (2FA) system based on the **D2OTP (Double Dynamic OTP)** protocol, which introduces an innovative approach to improve security through ephemeral keys and mutual authentication, without relying on asymmetric cryptography or PKI infrastructure.

The project involves the complete development of a 2FA solution, including a **RESTful API** built with **FastAPI**, a **cross-platform authenticator mobile application** developed with **Kotlin Multiplatform Mobile (KMM)**, and a **cloud infrastructure based on Firebase** for storage and messaging. The system verifies sensitive operations through an encrypted alternative channel, using the D2OTP protocol, which ensures the confidentiality, integrity, and authenticity of messages exchanged between client and server.

A **D2OTP cryptographic library** was implemented in both Python (for the backend) and Kotlin (for the mobile app), enabling the encoding and decoding of OTP messages with ephemeral keys. The D2OTP protocol avoids the reuse of secrets, prevents phishing and man-in-the-middle attacks, and complies with regulations such as the PSD2 directive in terms of strong customer authentication and dynamic linking.

The system was subjected to a thorough analysis of functional and non-functional requirements, as well as a threat model based on STRIDE and OWASP Mobile Top 10. Multiple risks associated with identity spoofing, information disclosure, data tampering, and privilege escalation were identified and mitigated, establishing a solid framework for comprehensive protection.

The implementation includes a modular and scalable architecture composed of the following components: a client app that initiates the authentication request, a backend that orchestrates the verification flow, and a 2FA app that allows the user to approve or reject the operation. Communication is carried out through encrypted messages (M1, M2, M3) following the D2OTP-defined flow, reinforced by an alternate channel (push or SMS).

Although the system is functional and robust, some limitations were identified, such as the incomplete integration of push notifications and the absence of recovery mechanisms in case of loss of the authenticator device. These shortcomings do not compromise the system's security but represent improvement areas for future versions.

In conclusion, this project demonstrates the feasibility and benefits of applying the D2OTP protocol as a second-factor solution in contexts that require high security but aim to avoid the complexity and cost of systems based on asymmetric cryptography. The solution developed provides a solid foundation for future research and deployment of advanced, adaptable authentication systems centered on user experience.

Contenido

1.Introducción

En la actual era digital, la seguridad informática y la salvaguarda de la información personal han cobrado una importancia nunca antes vista por el aumento exponencial en el uso de servicios digitales. El rápido progreso tecnológico y la digitalización de actividades diarias como las operaciones bancarias, las compras en línea, el acceso a servicios del gobierno y el trabajo a distancia han ocasionado que cada vez más datos delicados se guarden y se distribuyan de forma digital. Esto conlleva incrementos en los riesgos asociados a la seguridad y privacidad de la información personal, además de un incremento considerable de ciberataques que intentan aprovechar las vulnerabilidades en sistemas de computación y aplicaciones en línea.

El phishing, el ransomware, los ataques "man-in-the-middle", y el hurto de identidad son algunos de los riesgos más comunes y peligrosos que impactan a usuarios y empresas. De acuerdo con varios reportes de ciberseguridad, estos ataques han crecido significativamente en años recientes, provocando significativas pérdidas financieras y perjuicios a la reputación de las entidades impactadas. Esta situación obliga a reconsiderar las estrategias de seguridad digital que se suelen emplear y a buscar soluciones que brinden mayores garantías de resguardo frente a futuras amenazas cibernéticas.

La autenticación multifactor (MFA) es uno de los instrumentos más eficaces para salvaguardar la integridad y privacidad de la información personal. Este sistema incorpora niveles extra de protección al exigir varios procedimientos de verificación para verificar la identidad del usuario antes de autorizar el acceso a recursos resguardados. A pesar de su popularidad, muchas de estas soluciones poseen fallos de seguridad que pueden ser utilizados por un atacante, especialmente mediante el uso de métodos de autenticación estáticos o vulnerables como contraseñas simples, tokens enviados por mensaje de texto o autenticación basada en factores de reproducción simples.

Para enfrentar este problema, se introduce el protocolo Doble Dinámica OTP (D2OTP), una patente revolucionaria que ofrece un método sólido para la autenticación de múltiples factores. Este protocolo se distingue por crear claves únicas encriptadas de uso limitado para cada comunicación, proporcionando un incremento en la protección contra ataques convencionales y resistencia ante métodos avanzados de interceptación y gestión de datos. Su carácter adaptable y la utilización de autenticación recíproca entre cliente y servidor reducen considerablemente las posibilidades de ataques.

Este Trabajo de Fin de Grado se enfoca en la puesta en marcha de una solución fundamentada en D2OTP, tratando la construcción del backend a través de una API sólida y escalable desarrollada con FastAPI, así como el diseño e implementación de una aplicación móvil segura, eficaz y fácil de usar a través de Kotlin Multiplataforma Mobile (KMM) para utilizar dicho backend.

Objetivos del proyecto:

- Desarrollar una API robusta y adaptable que permita realizar sistemas 2FA a través del protocolo D2OTP.
- Desarrollar una aplicación móvil robusta, multiplataforma y orientada al usuario, que simplifica y proteja la gestión de la autenticación multifactor de forma segura y sencilla. Escoger e incorporar una biblioteca de cifrado sofisticada para llevar a cabo el encriptado seguro de claves efímeras, asegurando la privacidad e integridad de las comunicaciones.
- Ofrecer una infraestructura de nube segura, escalable y altamente accesible a través de Firebase para albergar tanto la API como los servicios relacionados.
- Llevar a cabo evaluaciones rigurosas para confirmar la seguridad, eficacia y escalabilidad del sistema integrado (backend, aplicación móvil y codificado).
- Analizar el efecto de la solución aplicada en aspectos de seguridad digital, acatamiento de regulaciones y sostenibilidad en el ámbito económico, social y ambiental.

Este documento se estructura en varios capítulos claramente definidos. Después de esta introducción, el Capítulo 2 aborda el estado del arte y trabajos previos relacionados con la seguridad digital, autenticación multifactor y protocolos criptográficos avanzados. El Capítulo 3 expone el progreso técnico llevado a cabo, especificando elementos fundamentales como la selección de tecnología, la creación de la API, la infraestructura y la aplicación para móviles. Luego, en el Capítulo 4 se muestran los logros alcanzados y las conclusiones que se extraen del proyecto. Finalmente, en el Capítulo 5 se presenta un análisis detallado del potencial impacto de la solución propuesta en diferentes campos y se proponen estrategias laborales futuras que permitan seguir reforzando la seguridad digital a raíz de este avance.
El propósito de este TFG es proporcionar una solución tecnológica sofisticada y segura que aporte significativamente a la salvaguarda de la información digital, respondiendo de manera eficiente a las amenazas en ascenso y estableciendo los fundamentos para futuras innovaciones en seguridad de la información.

2.Estado del arte (trabajos previos)

La autenticación en dos factores (2FA) se ha convertido en la salvaguarda esencial frente al robo masivo de credenciales. A lo largo de los últimos quince años hemos pasado de los clásicos tokens OTP hardware a aplicaciones móviles, y más recientemente a estándares basados en criptografía de clave pública. En esta sección se revisan sucintamente los pilares del 2FA (HOTP/TOTP), las aplicaciones comerciales que los popularizaron, los modelos *out-of-band*, la familia FIDO2/WebAuthn y, finalmente, el protocolo **D2OTP**, eje del presente Trabajo Fin de Grado. El objetivo es situar D2OTP dentro del panorama actual y resaltar sus aportaciones en autenticación mutua y uso de un canal dual.

2.1 OTP basados en algoritmos OATH (HOTP y TOTP)

El estándar **HOTP** (RFC 4226, 2005) define una contraseña de un solo uso derivada de una clave compartida y un contador creciente aplicados a un HMAC-SHA-1 [1]. El servidor y el cliente almacenan la misma clave simétrica; cada vez que el contador avanza ambos calculan el código y lo comparan. Para evitar problemas de desincronización y depender menos del usuario, la iniciativa OATH publicó en 2011 **TOTP** (RFC 6238), que reemplaza el contador por el tiempo: cada 30 segundos se genera un código nuevo usando la misma clave secreta y el valor de tiempo truncado [2]. De este modo no es necesario transmitir nada entre cliente y servidor salvo el OTP visible al usuario.

Los algoritmos OATH se han impuesto por su **simplicidad, costo nulo de licencia y cálculo local offline**: basta un reloj razonablemente sincronizado. Sin embargo, comparten vulnerabilidades intrínsecas: (i) los códigos se introducen manualmente, de modo que pueden divulgarse por error o phishing en tiempo real [3]; (ii) la clave secreta existe en dos lugares (servidor y dispositivo), de modo que su filtración permitiría generar OTP ilimitados; y (iii) la autenticación es **unidireccional**: sólo el servidor verifica al usuario, mientras que éste no puede confirmar la legitimidad del servicio. Ataques *man-in-the-middle* capaces de retransmitir un OTP en la misma ventana de 30 segundos han demostrado romper la barrera añadida por TOTP cuando la sesión web se encuentra secuestrada en vivo [4].

2.2 Aplicaciones comerciales que implementan OTP

La adopción masiva de 2FA llegó con los **autenticadores de software**. Google Authenticator —lanzado en 2010— utiliza TOTP/HOTP y permite que cualquier servicio publique un *QR code* con el secreto inicial. Una vez registrado, el usuario genera sus códigos sin conexión —ventaja clara frente al SMS— y sin coste de hardware. Alternativas como **Authy** incorporan

sincronización cifrada en la nube, recuperación en nuevos dispositivos y notificaciones *push* opcionales; **Microsoft Authenticator** añade a su vez la extensión "aprobar/rechazar" mediante *push* para cuentas corporativas, sustituyendo la introducción del código por un toque en pantalla. Estas aplicaciones han reducido el coste de despliegue y el rozamiento para el usuario, pero mantienen las limitaciones fundamentales de los OTP: un código revelado funciona mientras no caduque; el usuario no tiene forma de verificar el destino legítimo; y el canal puede ser imitado por un atacante de ingeniería social.

2.3 Autenticación *out-of-band* y doble canal

Para atajar el riesgo de interceptar el mismo flujo de datos, numerosos sistemas adoptaron la autenticación **fuera de banda** (OOB). Tras que el usuario introduzca sus credenciales en la web, el servicio envía un reto por un canal independiente —habitualmente la red móvil (SMS o llamada) o una *push notification*— que el usuario debe confirmar [5]. El atacante necesitaría comprometer ambos canales. La directiva PSD2 europea exige incluso que las transacciones de pago incorporen "información dinámica" firmada a través de un segundo canal, de modo que el usuario vea el importe y destino reales antes de aprobar [8].

Con todo, los canales OOB también presentan debilidades. La red telefónica ha mostrado vulnerabilidades (SS7, duplicado de SIM), motivo por el que NIST propuso **desaconsejar SMS** como segundo factor en su SP 800-63-3 draft de 2016 [5]. Las *push notifications* pueden verse afectadas por ataques de fatiga —bombardeo de solicitudes hasta que el usuario pulsa "Aceptar"— y siguen dependiendo de que la aplicación confíe en el servidor sin verificarlo de forma criptográfica. Por tanto, aunque el doble canal OOB reduce la superficie de ataque, el eslabón humano sigue siendo crítico.

2.4 Autenticación resistente al phishing: FIDO2 / WebAuthn

La **FIDO Alliance** y el W3C estandarizaron en 2019 la API **WebAuthn**, núcleo de la familia **FIDO2** [6]. El paradigma cambia: el servidor ya no almacena ni solicita secretos reutilizables, sino que conserva la **clave pública** de un par generado en un autenticador (llave de seguridad USB, módulo TPM, Secure Enclave, etc.). En el registro, el dispositivo crea la clave privada y devuelve la pública; en los inicios de sesión, firma un desafío específico del dominio y la sesión. El resultado es **resistencia nativa al phishing**: aun si el usuario interactúa con un sitio impostor, la firma no es válida para el dominio real, y la clave privada nunca abandona el dispositivo. Asimismo, el factor de posesión (la llave) puede protegerse con biometría o PIN local, logrando MFA o incluso experiencias *passwordless*. Grandes plataformas (Windows Hello, Android, iOS, navegadores modernos) soportan FIDO2, pero su adopción

universal se ve frenada por costes, disponibilidad de hardware y retrocompatibilidad con servicios existentes [7]. Mientras tanto conviven con OTP tradicionales.

2.5 Protocolo D2OTP: claves OTP efímeras y autenticación mutua

El **sistema D2OTP** (patente WO 2022/018310) se posiciona como una evolución intermedia —combina la sencillez de las OTP simétricas con propiedades de autenticación mutua típicas de la criptografía asimétrica— sin requerir infraestructura PKI ni dispositivos especializados [8]. Su operación puede resumirse en cuatro rasgos:

1. **Clave OTP por mensaje**: cada intercambio genera una clave de cifrado efímera; concluido el mensaje, la clave se desecha, logrando confidencialidad y anulando ataques de repetición.

2. **Autenticación mutua**: la respuesta del destinatario debe incorporar valores pseudoaleatorios enviados cifrados en el mensaje original; sólo quien descifre correctamente puede devolverlos, por lo que ambos extremos se autentican entre sí sin terceros.

3. **Canal dual lógico**: los valores de autenticación (tokens) viajan cifrados en el cuerpo del mensaje, de modo que romper la confidencialidad del canal de transporte no basta; además, en un despliegue de usuario-servidor la petición puede circular por Internet y la confirmación por un servicio *push* o SMS, conformando un doble canal físico.

4. **Rotación continua del secreto base**: tras cada diálogo correcto, parte de la información efímera se usa para derivar la clave base del próximo diálogo; así la "clave fija" compartida evoluciona y se convierte en una **clave de un solo uso ampliada** (One-Time Key), mejorando resistencia a fuerza bruta o leaks parciales.

2.5.1 Flujo resumido de autenticación

1. **Registro** (única vez): ambas aplicaciones comparten de forma segura un identificador y una clave primaria CC.

2. **Mensaje M1** (parte A → B): A cifra con CC datos + nonces (CCM2, VAM2, …) y envía.

3. **Mensaje M2** (B → A): B descifra con CC, valida hash, guarda los nonces y responde cifrando su carga con CCM2, incorporando uno de los nonces como prueba.
4. **Mensaje M3** (A → B): A descifra con CCM2, verifica el nonce para autenticar a B y confirma recepción cifrando con CCM3.
   Tras M3 ambas partes conocen que el diálogo fue íntegro, que los mensajes llegaron y que las claves efímeras se usaron correctamente. Si se requiere confirmación adicional B puede enviar M4, etc. En cada nuevo diálogo CC se sustituye por la derivada del anterior, convirtiéndose en un OTP de largo plazo.

2.5.2 Ventajas frente al estado del arte

- **Ausencia de secretos reutilizables en tránsito**: igual que FIDO2, nunca se expone la clave base; a diferencia de FIDO2, la solución es completamente simétrica y carece de dependencia de infraestructura de clave pública o llaves hardware.
- **Resistencia al phishing y MITM**: para que un atacante suplante a B necesita descifrar M1 (imposible sin CC) y generar M2 válido; para que suplante a A debe mostrar a B un M1 cuyos tokens se alineen con la vista de B, lo que tampoco puede lograr. De igual modo, retransmitir códigos no sirve: la clave OTP es distinta en cada mensaje y va cifrada.
- **Cumplimiento de requisitos regulatorios (PSD2 / SCA)**: la confirmación mutua y el vínculo entre mensajes pueden transportar datos dinámicos de la operación, satisfaciendo la exigencia de “firma dinámica” sin certificados [9].
- **Facilidad de despliegue**: basta integrar la librería D2OTP en el back-end (por ejemplo, la API FastAPI desarrollada en este TFG) y en la app móvil (KMM / Kotlin Multiplatform), apoyándose en Firebase para mensajería; no se precisan llaves U2F ni gestores de certificados.
- **Escalabilidad y modo “parte central”**: la patente contempla escenarios con una **parte central** (servidor) que mantiene las claves de todas las partes, simplificando la gestión en arquitecturas multi-cliente sin que los clientes tengan que almacenar secretos de terceros.

2.5.3 Comparación cualitativa

- Respecto a **TOTP/HOTP**: D2OTP elimina la entrada manual de códigos, proporciona autenticación mutua y usa claves efímeras; reduce la superficie de phishing al vincular cada token al diálogo específico.
- Frente a **OOB SMS/Push**: D2OTP ofrece verificación criptográfica integrada; el segundo canal es lógico o físico, pero no confía en la seguridad de la red móvil.
- En relación con **FIDO2/WebAuthn**: comparte resistencia a phishing y no repudio, pero emplea criptografía simétrica y puede funcionar en dispositivos que no soportan U2F ni passkeys; resulta una alternativa viable en casos donde la infraestructura FIDO no es adaptable.

2.6 Conclusión

El recorrido del 2FA muestra una evolución clara: de OTP simples (HOTP/TOTP) a aproximaciones más sofisticadas y resistentes al phishing como FIDO2. Durante la transición, soluciones intermedias que refuercen la autenticación simétrica siguen siendo valiosas, sobre todo cuando se busca compatibilidad con dispositivos existentes y baja complejidad de despliegue. El protocolo **D2OTP** reúne las ventajas de los OTP (ligereza, cálculos locales) y aporta autenticación mutua, claves efímeras y un concepto de canal dual que mitiga los ataques actuales contra 2FA basados en ingeniería social. Su adopción podría mejorar la seguridad en entornos donde FIDO2 todavía no es factible o donde se prefiera una solución OTP evolucionada sin dependencia de hardware. Estos factores justifican la investigación y la implementación realizada en el presente TFG.

3.Análisis de requisitos y modelo de amenazas

3.1 Requisitos funcionales y no funcionales

El sistema de autenticación de segundo factor (2FA) propuesto se integrará en un entorno de demostración compuesto por una aplicación móvil de servicio **cliente** (simulada como "banca falsa") desarrollada con Kotlin Multiplatform Mobile (KMM), una aplicación **autenticadora** en el dispositivo del usuario (también en KMM), un **backend** implementado con FastAPI (Python) que expone una API de verificación, y una base de datos en **Firebase Firestore** para almacenar información de usuarios, dispositivos y claves. La comunicación entre las aplicaciones móviles y el backend está protegida mediante el protocolo **D2OTP**, el cual introduce un esquema de cifrado simétrico de **OTP de un solo uso** y autenticación mutua en cada mensaje[10][11]. A continuación, se enumeran los requisitos del sistema, tanto funcionales como no funcionales, tomando en cuenta las necesidades del proyecto y las mejores prácticas en seguridad.

Requisitos Funcionales:

1. **Autenticación en dos pasos para operaciones:** El sistema debe permitir que la aplicación cliente bancaria solicite una **verificación de segundo factor** para operaciones sensibles (p. ej. inicio de sesión), enviando una petición que puede ser gestionada por el dispositivo autenticador. La solicitud incluirá datos relevantes de la operación a validar (ej. App que manda solicitud, tipo de solicitud) para que el usuario pueda tomar una decisión informada.
2. **Aprobación/Rechazo por el usuario:** La aplicación autenticadora móvil debe presentar al usuario la solicitud de autorización recibida y permitirle **aprobar o rechazar** la operación. Si el usuario no reconoce la solicitud, puede denegarla, protegiendo la cuenta frente a intentos de acceso no autorizados.
3. **Confirmación de la verificación:** Tras recibir la respuesta del usuario, el sistema backend debe procesarla y actualizar el estado de la solicitud en la Firebase Firestore. De este modo, el usuario puede darle a un botón de comprobar desde la app cliente que actuará en base al nuevo estado de la solicitud (p. ej. "Operación aprobada" o "Operación denegada"), cerrando el ciclo de verificación.
4. **Registro y gestión de dispositivos:** Debe implementarse un proceso seguro de **alta de nuevos dispositivos** autenticadores. Cada usuario podrá vincular un dispositivo móvil de confianza, que quedará registrado en el sistema junto con las claves necesarias para D2OTP. El alta de dispositivo seguirá el principio de "**trust on first use**" (confianza en el primer uso) para establecer una clave compartida inicial entre backend y dispositivo de forma segura durante el emparejamiento [16].

Asimismo, el sistema debe permitir cambiar el dispositivo vinculado a cada cuenta, garantizando que solo un dispositivo autorizado pueda gestionar respuestas 2FA.

5. **API de verificación 2FA:** El backend debe ofrecer una **API REST** segura que permita a las aplicaciones clientes (ej. la banca falsa) iniciar de manera asíncrona las solicitudes de segundo factor. Esta API recibirá la identificación de la operación/crear_solicitud junto con un mensaje M1 que descifrará y verificará para obtener los datos de la solicitud que va a crear. De esta forma, el esquema es extensible a múltiples aplicaciones clientes que requieran 2FA, abstrayendo la lógica de verificación en un servicio central.
6. **Canal de comunicación alternativo:** La transmisión de los mensajes de verificación debe realizarse por un **canal independiente** del canal principal de la operación, reforzando la seguridad frente a atacantes en la red. Se evaluará el uso de **SMS** para enviar los mensajes D2OTP (M1, M2, M3) como canal out-of-band, aunque también se contemplan alternativas como notificaciones **PUSH** (Firebase Cloud Messaging) o incluso peticiones REST directas en caso de que SMS no resulte viable o seguro. Esta flexibilidad en el canal de transporte busca garantizar la entrega de los códigos de un solo uso incluso bajo distintas condiciones de conectividad del usuario, a la vez que dificulta que un atacante pueda interceptar ambos factores (credenciales y OTP) por un mismo medio.
7. **Librería cliente D2OTP:** Debe desarrollarse o integrarse una **biblioteca criptográfica** que implemente el protocolo D2OTP. Esta librería se incorporará tanto en el backend como en las aplicaciones móviles, proporcionando funciones para **codificar y decodificar** los mensajes (M1, M2, M3) con cifrado OTP y verificaciones de autenticidad. Gracias a ello, se asegura la interoperabilidad y correcta ejecución del protocolo, evitando errores de implementación. La librería deberá exponer interfaces sencillas para que los desarrolladores de la app cliente puedan invocar el envío de solicitudes 2FA sin profundizar en los detalles de bajo nivel del cifrado.
8. **Notificación al usuario y experiencia integrada:** En la aplicación autenticadora, las solicitudes entrantes de 2FA deberían generar una notificación instantánea (ej. mediante los mecanismos de notificación del sistema operativo) para alertar al usuario de que hay una operación pendiente de confirmar. Asimismo, la interfaz debe ser mínima y clara (mostrar información de la operación y botones de aprobar/denegar) para agilizar la respuesta.

Requisitos No Funcionales:

- **Seguridad criptográfica (Confidencialidad e Integridad):** La comunicación entre el backend y las apps móviles debe garantizar la **confidencialidad** de las credenciales de un solo uso y la **integridad** de los mensajes intercambiados, incluso si viajan por redes inseguras o públicas [10]. Esto implica que ningún tercero pueda leer o modificar los mensajes M1, M2, M3 en tránsito. Dado que D2OTP cifra cada

mensaje con una clave *otp* de un único uso, se cumple este requisito evitando la reutilización de claves de cifrado en distintos mensajes[10]. Adicionalmente, se usará TLS/HTTPS en las comunicaciones API REST cuando sea posible, añadiendo una capa extra de cifrado durante el transporte [12].

- **Autenticación mutua de extremos:** El diseño debe proporcionar **verificación mutua** tanto del servidor como del dispositivo cliente en cada intercambio de mensajes, de modo que cada parte confirme la identidad de la otra. Este requisito busca impedir ataques de **phishing** o suplantación, en los cuales un atacante podría intentar hacerse pasar por el servidor legítimo ante el usuario o viceversa. El protocolo D2OTP aborda esto incorporando autenticación mutua en cada comunicación, lo que previene ataques de *Man-in-the-Middle* al exigir prueba criptográfica de la identidad de emisor y receptor[10][11]. Gracias a ello, el usuario solo aprobará solicitudes que provienen realmente de su banco, y el backend solo aceptará respuestas firmadas por el dispositivo registrado del usuario.
- **No repudio y trazabilidad:** Cada evento de autenticación debe generar evidencias que impidan el **repudio** por las partes involucradas. Es decir, ni el usuario debería poder negar que aprobó una operación, ni el servicio negar que envió la petición, en caso de disputas. Para lograrlo, cada mensaje intercambiado en D2OTP lleva asociados mecanismos de verificación (p.ej. MAC o hash con claves OTP) que sirven como prueba criptográfica de la participación de cada extremo[10]. Además, se recomienda registrar en bitácoras seguras cada solicitud de 2FA, su aprobación/denegación y los identificadores de mensaje o *nonces* utilizados, respetando la privacidad. D2OTP facilita esta trazabilidad intrínseca: cada interacción deja un registro verificable sin necesidad de servidores de log centralizados[11]. Este no repudio técnico resulta especialmente relevante en entornos financieros o de pago electrónico, alineándose con exigencias regulatorias de mantener evidencias de las transacciones autenticadas[11].
- **Cumplimiento de normativas (PSD2 y similares):** El sistema deberá apoyar el **cumplimiento normativo** en contextos donde la autenticación reforzada es obligatoria. Por ejemplo, la normativa europea PSD2 exige **Strong Customer Authentication (SCA)** para operaciones bancarias, lo que implica dos factores de distinta categoría y la "**vinculación dinámica**" de la autenticación con los datos de la transacción (monto y beneficiario)[10]. La solución propuesta, al proveer un segundo factor robusto y permitir incluir en el mensaje de verificación información contextual de la operación, contribuye a satisfacer estos requerimientos. Además, el uso de OTP únicos por transacción actúa como código de enlace que ata la aprobación del usuario a esa transacción específica, dificultando ataques de reutilización o aprobación genérica. Se considerarán también lineamientos de estándares como FIDO2/WebAuthn y recomendaciones de organismos de ciberseguridad (OWASP, INCIBE, CCN-CERT) para

asegurar que la implementación sigue las mejores prácticas reconocidas.

- **Rendimiento y escalabilidad:** La adición del segundo factor no debe introducir demoras excesivas en la experiencia de autenticación. Se requiere que el **tiempo de verificación** (desde que el usuario inicia sesión hasta que se valida el 2FA) sea razonablemente corto (idealmente unos pocos segundos). D2OTP está optimizado en este sentido, pues utiliza cifrado **simétrico** ligero en vez de operaciones de clave pública costosas, reduciendo la latencia de procesamiento[10]. Además, al no depender de terceras autoridades ni realizar *handshakes* PKI en cada sesión, se minimiza el número de rondas de comunicación necesarias[10]. En cuanto a **escalabilidad**, el sistema debe soportar incrementos en la cantidad de usuarios, dispositivos y transacciones sin pérdida significativa de rendimiento. El backend en FastAPI deberá ser capaz de manejar múltiples solicitudes concurrentes de 2FA, y la infraestructura (Firestore, servicio de mensajería) escalar según la demanda. Firestore aporta alta escalabilidad horizontal y rendimiento en accesos concurrentes, y cifra todos los datos almacenados en disco automáticamente [12], lo cual contribuye simultáneamente a seguridad y rendimiento.
- **Disponibilidad y tolerancia a fallos:** El servicio 2FA debe ser **altamente disponible**, ya que una caída implicaría bloquear accesos legítimos de usuarios. Se procurará desplegar el backend de forma redundante en producción; en este TFG, aunque el backend está alojado localmente y expuesto mediante **ngrok** (túnel), cambiando la Url en cada arranque del servicio, se asume para fines demostrativos. No obstante, se han de manejar adecuadamente posibles fallos de comunicación: por ejemplo, las operaciones de segundo factor tendrán un **tiempo de expiración** definido (p.ej. el OTP solo es válido por X minutos) para evitar que solicitudes antiguas queden abiertas indefinidamente.
- **Almacenamiento seguro de datos sensibles:** Todos los datos sensibles manejados por el sistema – como las claves compartidas entre backend y dispositivo (clave base *K_AB* de D2OTP), *nonces* OTP, tokens de sesión, etc. – deben almacenarse de forma segura. En el backend, Firestore cifra los datos en reposo y en tránsito [12], pero es crucial definir **reglas de seguridad Firebase** estrictas para que solo usuarios/autoservicios autorizados accedan a cada registro. Igualmente, la librería D2OTP no debe exponer las claves en memoria más de lo necesario. Este manejo seguro de credenciales mitiga riesgos del OWASP Mobile Top 10 como el **M2: Insecure Data Storage** [13].
- **Usabilidad y experiencia de usuario:** Finalmente, aunque la seguridad es prioritaria, el sistema debe ser **cómodo de usar**. Un 2FA demasiado engorroso puede provocar errores o desuso por parte de los usuarios. Se requiere que la interacción añadida (recibir una notificación o iniciar sesión, y pulsar “aprobar”) sea lo más fluida posible, minimizando fricciones. Según NIST, es recomendable evitar factores que el usuario

deba transcribir manualmente en la medida de lo posible [16], por lo que la solución de aprobación con un toque en una app dedicada resulta adecuada.

3.2 Modelo de amenazas (STRIDE y OWASP Mobile Top 10)

En esta sección se analiza el **modelo de amenazas** del sistema, identificando los riesgos principales que deben afrontarse en el diseño. Se ha adoptado una aproximación combinada: por un lado, se aplica la metodología **STRIDE** de Microsoft para categorizar las amenazas de seguridad en seis grandes clases (Suplantación, Manipulación, Repudio, Divulgación de información, Denegación de servicio, Elevación de privilegios) [14]. Por otro lado, se consideran las vulnerabilidades típicas de aplicaciones móviles recopiladas en el **OWASP Mobile Top 10 (versión 2016)** [13], para no pasar por alto debilidades comunes en este tipo de sistemas (almacenamiento inseguro, comunicaciones inseguras, autenticación rota, etc.). A continuación, se detallan las principales amenazas identificadas, indicando en cada caso su relación con las categorías STRIDE y, cuando corresponde, con los riesgos móviles de OWASP:

- **Suplantación de identidad (Spoofing):** Un atacante podría intentar **hacerse pasar por una entidad legítima** dentro del sistema. Esto incluye la suplantación de un usuario legítimo (obteniendo sus credenciales de primer factor y tratando de engañar al segundo factor) o la suplantación del servidor/autenticador en la comunicación. Por ejemplo, un atacante de tipo *phishing* podría enviar al usuario una notificación falsa solicitando aprobar un inicio de sesión, simulando ser la app bancaria. Si el usuario la acepta sin darse cuenta del engaño, el atacante obtendría acceso. Esta amenaza corresponde a la "S" de STRIDE [14]. En términos de OWASP, se relaciona con **M4: Insecure Authentication** [13], pues suele explotar debilidades en la verificación de identidad, y con **M9: Reverse Engineering**, en el sentido de que si la app no valida adecuadamente la identidad del backend (por ejemplo, no verifica certificados TLS o la firma de los mensajes), un tercero podría hacerse pasar por el servidor. **Mitigación:** Para contrarrestar esta amenaza, el sistema implementa autenticación mutua mediante D2OTP y asegura que cada mensaje M1/M2 vaya cifrado y firmado de forma que el receptor verifique criptográficamente el remitente legítimo[10]. Adicionalmente, el usuario debe revisar la información contextual de cada solicitud en la app autenticadora (p. ej. origen de la petición, datos de la operación) y solo aprobar si reconoce haberla iniciado, reduciendo la eficacia de intentos de spoofing.
- **Manipulación de datos (Tampering):** Se refiere a la **alteración maliciosa de los datos o del código** del sistema. Un adversario con capacidades en la red podría intentar interceptar y modificar el contenido de los mensajes M1, M2 o M3 en tránsito, alterando por ejemplo el código OTP o la respuesta del usuario (amenaza de integridad) [14]. Igualmente, un atacante local que obtenga acceso al

dispositivo móvil podría intentar manipular la aplicación autenticadora – por ejemplo, mediante técnicas de hooking o parcheo en memoria – para que autorice automáticamente todas las solicitudes (bypasseando la decisión del usuario). En el backend, un atacante podría tratar de manipular la base de datos (si encontrase una vulnerabilidad) para registrar su propio dispositivo como autenticador de la víctima. Esta categoría (la "T" de STRIDE) cubre un amplio espectro de ataques. En OWASP Top 10 Móvil, la manipulación se asocia principalmente a **M3: Insecure Communication** (si no hubiera cifrado, sería trivial alterar mensajes en tránsito) y **M8: Code Tampering** [13] (modificación del binario o comportamiento de la app cliente). **Mitigación:** El uso de **cifrado robusto end-to-end** en D2OTP garantiza que aunque un atacante intercepte los mensajes, no pueda descifrarlos ni modificarlos sin ser detectado (cualquier alteración invalidaría la autenticación del mensaje). Cada mensaje lleva integridad asegurada, por lo que la manipulación resulta en un rechazo de la comunicación. Además, la arquitectura con **claves OTP de un solo uso** dificulta que un atacante pueda reutilizar o inyectar mensajes antiguos: aun si lograra extraer un mensaje M1 y modificarlo, el protocolo exige valores aleatorios únicos por sesión, previniendo ataques de **replay o impersonación**[10][11].

- **Repudio (Repudiation):** El repudio se refiere a la posibilidad de que una de las partes niegue haber realizado una acción o transacción [14]. En nuestro contexto, un riesgo sería que un usuario malicioso apruebe una operación y luego reclame que su cuenta fue comprometida, o bien que el servidor niegue haber enviado cierta instrucción. Sin controles adecuados, esto podría generar disputas difíciles de resolver. Si no se registran evidencias, un usuario podría alegar "yo nunca aprobé esa transferencia" y sería complicado demostrar lo contrario. **Mitigación:** Esta amenaza (la "R" de STRIDE) se aborda diseñando el sistema de forma que haya **trazabilidad y evidencias criptográficas**. D2OTP, por su diseño, proporciona *evidencias de no repudio*: cada mensaje intercambiado está autenticado con claves que solo poseen los participantes legítimos, lo que significa que puede demostrarse que una respuesta M2 vino del dispositivo del usuario (porque solo él tenía la clave OTP para generarla correctamente)[10].
- **Divulgación de información (Information Disclosure):** Representa las amenazas asociadas a la **exposición no autorizada de información confidencial** [14]. En un sistema 2FA, los principales datos sensibles son las **claves secretas** (ej. la clave simétrica base *K_AB* que comparten dispositivo y servidor, o los identificadores de sesión), así como los códigos de un solo uso generados. Si un atacante obtuviese el secreto primario de un usuario, podría generar sus OTP y derrotar la autenticación de segundo factor. También la información personal de usuarios almacenada en Firestore, o los detalles de las operaciones en las notificaciones, entran en esta categoría. Desde la perspectiva OWASP, esto engloba **M2: Insecure Data Storage** (p. ej. almacenamiento de claves en el dispositivo sin cifrar) y **M3:**

Insecure Communication (exposición de datos en tránsito) [13], e incluso **M10: Extraneous Functionality** si existieran interfaces ocultas que filtren datos [13]. **Mitigación:** Se aplican varios controles para prevenir fugas de información. En tránsito, **toda comunicación va cifrada** con D2OTP (además de TLS en la API), asegurando confidencialidad de OTP y datos personales[10]. En reposo, Firebase Firestore **cifra automáticamente** los datos antes de almacenarlos en disco y mientras están en tránsito hacia el cliente [12]. Pero más importante aún, se configuran reglas de acceso para que solo los usuarios propietarios puedan leer los datos de sus operaciones y claves públicas de dispositivos. Adicionalmente, se deben desechar correctamente de memoria los datos sensibles tras su uso para que no queden expuestos a nivel de RAM. Cabe notar que, al utilizar D2OTP, **no se almacena un secreto estático** que genere todos los OTP (como ocurre en TOTP/HOTP); en su lugar, cada sesión utiliza OTP derivados y valores aleatorios. Esto limita el impacto de una posible divulgación: aun si se filtrase un OTP puntual, no sería reutilizable para futuras sesiones, y descubrir la clave base *K_AB* a partir de OTP pasados es computacionalmente inviable debido a la seguridad criptográfica del algoritmo[10].

- **Elevación de privilegios (Elevation of Privilege):** Esta última categoría de STRIDE cubre escenarios en los que un atacante con acceso limitado consigue **elevar sus privilegios** y obtener accesos no autorizados [14]. En el caso de la 2FA, podemos imaginar que un atacante logra comprometer parcialmente el sistema – por ejemplo, conoce la contraseña del usuario (primer factor) – e intentará escalar privilegios obteniendo también el segundo factor. Si el atacante controla el teléfono de la víctima (o una sesión de malware en él), podría intentar aprovechar esa posición para aprobar operaciones sin que el usuario lo sepa, convirtiéndose efectivamente en un usuario con privilegios elevados. Otro ejemplo sería que un atacante que comprometió el backend con privilegios bajos (digamos acceso a ciertas colecciones de Firestore) intente acceder a datos más sensibles o insertar claves falsas de dispositivo en la cuenta de un usuario (haciéndose añadir como dispositivo autenticador). **Mitigación:** Para contener esta amenaza, se implementan **controles de autorización estrictos** en todas las capas. En Firestore, las reglas se aseguran de que ninguna ajena a la API RESTful del backend pueda escribir en, evitando que un atacante "inyecte" un dispositivo no autorizado. Del lado cliente, incluso si un malware obtiene permisos de accesibilidad o root, la app autenticadora debería idealmente requerir una **autenticación local** (PIN o biométrica) para abrir la aplicación o aprobar solicitudes, de modo que no cualquier proceso automatizado pueda simular la interacción del usuario real. Esto emula las recomendaciones de nivel AAL3 de NIST, donde se pide un factor físico (ej. huella) para activación del segundo factor [16]. Asimismo, cualquier intento de manipular privilegios se vería dificultado por la mutualidad de D2OTP: aunque un atacante lograse acceder como usuario en el backend, sin poseer la clave simétrica del

dispositivo no podrá generar respuestas M2 válidas (no basta con "decirle al servidor que la 2FA pasó", debe proveer la prueba criptográfica). Del mismo modo, un atacante con control parcial del dispositivo pero que no conozca el PIN de desbloqueo o la biometría del usuario, no podría utilizar la app autenticadora si ésta lo exige para cada aprobación (este es un parámetro de diseño que podría incorporarse para operaciones de alto riesgo). En resumen, la arquitectura procura que **cada componente solo pueda realizar acciones acordes a su rol**, y que lograr un compromiso completo requiera múltiples fallos encadenados. Esto sigue el principle de *defensa en profundidad*: incluso si un control falla (ej. la contraseña fue robada), el atacante enfrentará nuevas barreras antes de lograr privilegios de aprobación de operaciones.

Cabe destacar que las amenazas descritas no son teóricas sino respaldadas por incidentes conocidos y estudios de seguridad. Por ejemplo, OWASP identifica el almacenamiento inseguro y las comunicaciones inseguras entre las principales causas de fugas de datos en apps móviles [13]. Asimismo, ataques de phishing avanzados intentan engañar al usuario para que autorice operaciones fraudulentas ("**approval fraud**"), lo cual ha llevado a instituciones a reforzar la contextualización de sus segundos factores[10][10]. El modelo de amenazas aquí elaborado sirve como base para definir las **contramedidas** presentadas en la siguiente sección, de modo que el diseño del sistema aborde proactivamente cada riesgo identificado.

3.3 Limitaciones y aspectos a reforzar

Si bien el sistema diseñado aporta notables mejoras de seguridad, durante el análisis se han identificado **limitaciones** y áreas que podrían requerir fortalecimiento en futuras iteraciones. Reconocer estas limitaciones es fundamental para contextualizar el nivel de seguridad alcanzado y entender en qué supuestos podría un adversario avanzar. A continuación, se detallan los principales aspectos a considerar:

- **Seguridad del dispositivo del usuario:** La solución asume que el dispositivo móvil del usuario (smartphone) es un entorno relativamente confiable para albergar la aplicación autenticadora. En la práctica, esta premisa puede fallar si el dispositivo está **comprometido por malware** o si el usuario lo ha rooteado/jailbreakeado rompiendo sandbox de seguridad. Un malware con altos privilegios en el teléfono podría, en teoría, capturar tanto la contraseña inicial (por keylogger o phishing en el dispositivo) como interceptar/aprobar automáticamente las solicitudes 2FA, anulando la efectividad del segundo factor. Este es un problema general con los 2FA basados en el *mismo dispositivo* del que parte la operación. Por ejemplo, un troyano bancario móvil podría superponerse a la app bancaria e igualmente leer las notificaciones de la app autenticadora, logrando fraude. **Implicación:** Aún con D2OTP, si el atacante controla el dispositivo, la autenticación queda comprometida. De hecho, los estándares de mayor nivel (ej. NIST AAL3) requieren factores con **resistencia a verificación por el verificador** y

preferiblemente basados en hardware separado (como una llave física) [16]. En nuestro caso, no se utiliza un elemento seguro hardware aparte del teléfono, por lo que **no se llega al nivel de seguridad de un token hardware FIDO2**. Esto es aceptable en un entorno de usuario promedio, pero para escenarios de altísima seguridad (p. ej. cuentas privilegiadas) habría que considerar complementar con un factor adicional o usar enclaves más fuertes. En resumen, la confianza sigue residiendo en gran medida en la **seguridad del endpoint**; educar al usuario para mantener su dispositivo libre de malware, actualizado y bloqueado es crítico.

- **Dependencia en SMS como canal (si se usa):** Aunque el diseño contempla alternativas, el uso de SMS para transmitir mensajes D2OTP fue evaluado por su ubiquidad. Sin embargo, SMS es reconocido como un canal vulnerable: suceptible a ataques de **SIM swapping**, intercepción en la red SS7, redirección de mensajes, etc. Diversos organismos desaconsejan SMS como segundo factor cuando se requiere alta seguridad. NIST, por ejemplo, en sus guías más recientes **desaprueba SMS** y canales de voz/Email para verificación en dos pasos, debido a estas vulnerabilidades inherentes [16]. En nuestro sistema, si se confía en SMS, un atacante que convenza a la compañía telefónica para transferir el número del usuario a otra SIM podría recibir los OTP (aunque estén cifrados, podría iniciar una sesión maliciosa con la víctima para obtenerlos en contexto válido). Asimismo, los SMS pueden demorarse o perderse, afectando la usabilidad. **Mitigación/Contramedida:** Se sugiere privilegiar las notificaciones push cifradas o canales IP seguros por encima de SMS siempre que sea posible. Push, combinado con D2OTP, ofrece un grado de seguridad muy alto (capa TLS + cifrado OTP, difícil de interceptar). Si por requisitos de compatibilidad se mantiene SMS, habría que reforzar la **verificación de cambio de SIM**: por ejemplo, invalidar dispositivos 2FA cuando se detecte cambio de número u ofrecer al usuario alertas out-of-band ante cambios. Otra estrategia es enviar mensajes M1 *vacíos* o sin datos sensibles por SMS y obligar a la app a solicitar por canal seguro los detalles (minimizando lo que podría exponerse vía telecom). En cualquier caso, para entornos que lo permitan, migrar a un modelo estilo **"push-to-verify"** sería deseable.
- **Confiabilidad del enlace ngrok/local:** Durante el desarrollo del TFG, el backend se alojó en local y se utilizó **ngrok** para exponerlo públicamente. Si bien esto es útil para demostraciones, en producción no es recomendable por temas de fiabilidad y control. Ngrok introduce una dependencia en un servicio tercero para el túnel; si ese servicio falla, las autenticaciones fallarían. Además, el dominio generado es aleatorio y podría complicar la validación estricta de origen en la app (aunque se puede fijar un subdominio de pago). Por otro lado, el servidor local puede no tener las mismas garantías de seguridad física y de red que un despliegue en la nube o data center. **Recomendación:** Planificar la migración del backend a un entorno más robusto (por ejemplo, una instancia cloud con certificación TLS propia) y utilizar un

nombre de dominio fijo para las conexiones de la app. Esto permitirá también implementar **pinning** del certificado o al menos de la identidad del servidor en la aplicación móvil, evitando ataques de personificación del servidor 2FA. Actualmente, si bien D2OTP no depende de TLS para la confidencialidad, sí se apoya en la autenticación mutua interna; sin embargo, agregar pinning TLS brindaría defensa adicional contra un MITM que intente interponerse antes de establecer siquiera D2OTP. Esta mejora refuerza la protección contra **Spoofing de servidor** incluso en el arranque de la comunicación.

- **Complejidad y adopción de D2OTP (algoritmo propietario):** D2OTP es un protocolo patentado e innovador, pero justamente por ser novedoso carece del grado de **escrutinio público** que tienen estándares maduros (como TLS, OATH TOTP, etc.). La comunidad de seguridad suele recomendar precaución con algoritmos criptográficos propietarios o poco analizados [15]. Si bien D2OTP en papel ofrece excelentes propiedades de seguridad, en la implementación real podrían surgir vulnerabilidades si no se realiza correctamente o si la patente omitió algún vector (por ejemplo, consideraciones de sincronización, o generación de aleatorios débiles). La limitada adopción hasta la fecha implica que herramientas de prueba de penetración comunes no lo han tenido como objetivo ampliamente, pudiendo haber *zero-days* desconocidos. **Consideración:** Para contrarrestar esto, sería conveniente **someter D2OTP a auditorías externas** o compararlo con esquemas estándar equivalentes. En entornos donde la conformidad con estándares sea obligatoria, quizás se deba ofrecer como alternativa métodos de 2FA más tradicionales configurables (ej: TOTP RFC 6238). No obstante, cabe mencionar que D2OTP está alineado con principios de seguridad sólidos (simetría, OTP, mutua autenticación) y que incluso se postula como resistente a ataques cuánticos al no usar asimetría [11]. Aun así, la prudencia indica monitorizar su desarrollo. La **interoperabilidad** es otra arista: al ser propietario, otras plataformas no tienen implementaciones nativas; por tanto, la librería desarrollada necesita mantenerse actualizada y podría requerir licenciamiento para usos comerciales. Esto podría frenar la adopción masiva si no se maneja con una estrategia abierta o de estandarización a futuro.
- **Protección contra factores humanos:** Ninguna solución tecnológica es totalmente inmune a errores o engaños humanos. Un usuario distraído podría aprobar una solicitud de 2FA sin revisarla, incluso con todas las medidas implementadas (social engineering puro). Nuestro sistema mitiga mucho el phishing técnico, pero **educar al usuario** sigue siendo vital: deben entender que nunca aprobarán una petición que no esperaban. Asimismo, debe haber procesos en caso de dispositivo perdido/robado: si un atacante roba el móvil desbloqueado de la víctima, podría acceder tanto a la banca como a la app autenticadora. En ese caso, el 2FA no impide el fraude porque ambos factores estaban físicamente juntos comprometidos. **Medidas organizativas:** Se recomienda implementar un sistema de claves de recuperación en el registro. De esa manera, si un atacante enrolara un dispositivo falso (lo

cual debería ser muy difícil por las claves, pero imaginemos que ocurrió vía acceso al Firestore), el usuario legítimo podría recibir una alerta y reaccionar. También se debería instruir a los usuarios a mantener actualizadas las apps, ya que futuras vulnerabilidades descubiertas necesitarán parche (por ejemplo, si se hallara un bug en la generación de OTP). Un aspecto a reforzar es la **UX de error**: si una petición expira o es rechazada, la app cliente podría informar "Tiempo de verificación agotado" en lugar de simplemente fallar, para que el usuario no desarrolle el hábito de reenviar múltiples veces (lo cual podría ser explotado por un atacante para confundirlo).

En conclusión, las limitaciones mencionadas no representan fallos críticos del diseño sino **consideraciones para la mejora continua**. La seguridad es un proceso incremental: el sistema planteado eleva significativamente la barra frente a esquemas 2FA tradicionales (especialmente en prevención de phishing, MITM y replay). No obstante, siempre existirán escenarios extremos o vectores avanzados a contemplar. Reforzar la seguridad del dispositivo cliente, minimizar la dependencia de canales débiles como SMS [16], y mantener las implementaciones alineadas con estándares reconocidos serán pasos lógicos a seguir. Pese a estas áreas de atención, el modelo implementado ofrece un **balance favorable** entre seguridad y usabilidad, cumpliendo los objetivos del proyecto de proporcionar una capa de autenticación adicional robusta y adaptable a diferentes aplicaciones. Cada limitación aquí identificada puede ser abordada con mitigaciones adicionales, lo que demuestra la flexibilidad del sistema para evolucionar ante futuros requerimientos o amenazas emergentes.

4.Diseño de la arquitectura

En esta sección abordaremos el diseño de la arquitectura del sistema, con la finalidad de ayudar a su implementación con otras aplicaciones existentes.

4.1 Objetivos de diseño y requisitos de alto nivel

En esta sección definiremos los proncipios seguidos a la hora de desarrollar el sistema de autenticación 2FA basado en D2OTP y los requisitos de cada componente.

ID	Objetivo	Descripción resumida	Referencias
O-1	**Seguridad extremo a extremo**	Garantizar confidencialidad, integridad, autenticación mutua y no repudio en todos los mensajes M1–M3 mediante claves OTP de un solo uso CCAB.	Patente D2OTP [8] NIST SP 800-63B [17]
O-2	**Modularidad y reutilización**	Diseñar la solución como un módulo 2FA modular que pueda integrarse en distintas aplicaciones cliente sin cambios profundos.	Plan de Trabajo [Anexo 1]
O-3	**Escalabilidad horizontal**	Soportar incrementos de usuarios y operaciones usando servicios gestionados (Firebase, FCM) y un backend asíncrono FastAPI.	Google Cloud Firestore [18] FastAPI Docs [19]
O-4	**Compatibilidad multiplataforma**	Compartir lógica con KMM; concentrar el MVP en Android, manteniendo la posibilidad de portar a iOS en el futuro.	JetBrains KMM Guide [20]
O-5	**Cumplimiento normativo (PSD2-SCA)**	Proveer doble factor independiente y vinculación dinámica con los datos de la operación, conforme a RTS PSD2.	EBA RTS SCA [9]
O-6	**Observabilidad y auditoría**	Registrar eventos críticos (registro de dispositivo, M1, M2, M3) con marcas de tiempo para análisis forense y no repudio.	OWASP MASVS v2 [21]

4.1.1 Requisitos de alto nivel

1. **R-F1 Gestión de dispositivos:**
    - Alta, revocación y modificación de dispositivo confiable por usuario.
    - Generación segura de *CCAB* durante el emparejamiento inicial.
    - Almacenamiento cifrado de claves en Firestore y en Android Keystore.
2. **R-F2 API de verificación asíncrona:**
    - Punto /crear-solicitud que acepte M1 compuesto de {userId, appId, txnData,...} y devuelva requestId (cifrado dentro de un M2).
3. **R-F3 Canales de transporte alternativos:**
    - Soportar al menos **FCM Push** y **REST**; SMS como opcional.
    - Resiliencia a fallos: reintento y expiración configurable (p. ej. 90 s).
4. **R-NF1 Latencia y rendimiento:**
    - Tiempo p95 < 200 ms en backend; ≤ 1 s procesamiento en app móvil.
    - Throughput estimado: 50 req/s en entorno de pruebas.
5. **R-NF2 Disponibilidad:**
    - SLA objetivo ≥ 99,5 % (entorno productivo).
    - Despliegue redundante aconsejado; en el TFG se simula con ngrok.
6. **R-NF3 Privacidad y protección de datos:**
    - Minimizar DCP: almacenar solo hashes de contraseña y metadata necesaria.
    - Cumplir principios GDPR (minimización, limitación de finalidad).

4.2 Vista de componentes

4.2.1 Descripción general

El sistema se concibe como una arquitectura **cliente-servidor distribuida** con un **canal alternativo cifrado** (D2OTP) para la verificación de segundo factor. Consta de:

1. **Aplicación bancaria (App Cliente)** — inician las operaciones sensibles y solicita 2FA.
2. **Backend API (FastAPI)** — expone endpoints REST y persiste estado.
3. **Aplicación autenticadora (App 2FA)** — consulta la petición, muestra los datos y envía la respuesta a backend, para más tarde ser verificada por Cliente.
4. **Servicios de soporte** —
    - **Cloud Firestore** para usuarios, peticiones y dispositivos (datos cifrados en reposo por defecto) [12].
    - **Firebase Cloud Messaging (FCM)** como canal push.
5. **Librería D2OTP** — código común (Kotlin + Python) que codifica/decodifica M1-M3.

La elección de **FastAPI** responde a su alto rendimiento asíncrono y tipado estático ligero [19]; la de **KMM** asegura lógica compartida, soporte desde Android Studio y futura portabilidad a iOS. Las vistas posteriores (despliegue y secuencia) se apoyarán sobre este inventario de componentes.

4.2.2 Componentes lógicos

App Cliente (KMM)

Es la app desde la que se pretende hacer un login. La idea del sistema de autenticación es asegurarse de que se pueda integrar fácilmente en este tipo de aplicaciones.

Backend API (FastAPI)

Encargada de gestionar la base de datos donde se encuentran las solicitudes, los usuarios, y las apps cliente (con sus respectivas claves de cifrado OTP CCAB). Es un servicio RESTful que recibe y envía todas sus comunicaciones siguiendo el protocolo D2OTP.

En la implementación actual, recibe mensajes M1 y M3, siempre que responde lo hace con M2.

Genera solicitud 2FA (M1).

• Envía M1 de push vía FCM/SMS.

• Valida M2 de respuesta cuando el dispositivo recibe la notificación.

App 2FA (KMM)

• Decodifica M1 (D2OTP).

• Muestra detalles y pide PIN/biometría opcional.

• Envía M2. FCM push / SMS intent; REST/JSON reply.

Cloud Firestore

• Colecciones: users, devices, requests, logs.

• Cifrado en reposo automático. gRPC/HTTPS SDK.

FCM / SMS Gateway

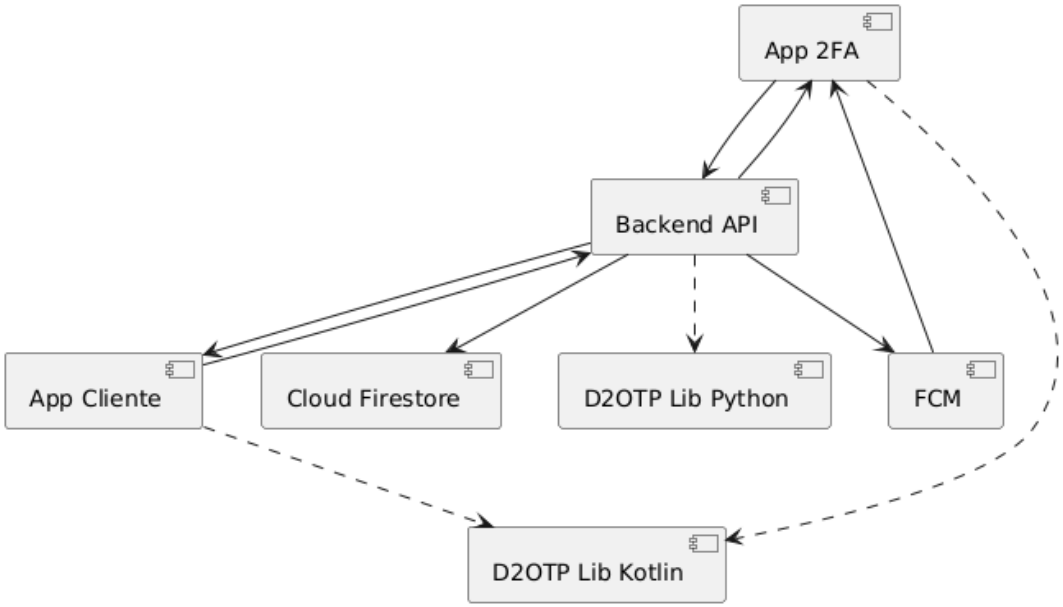
• Transporte out-of-band de M1. HTTP v1 (FCM); SMPP/REST (SMS).

Librería D2OTP

• Derivación de OTP, cifrado, MAC.

• Verificación mutua emisor/receptor. API local (Kotlin multiplatform / Python).

4.2.3 Diagrama de componentes



- **App Cliente:** app bancaria demo que inicia la operación y pide al backend la verificación 2FA.
- **Backend API:** servicio FastAPI que orquesta la autenticación: guarda la petición, envía la notificación y comprueba la respuesta.
- **Cloud Firestore:** base de datos NoSQL donde se registran usuarios, dispositivos, claves y solicitudes 2FA.
- **FCM:** Firebase Cloud Messaging, canal push que entrega la solicitud 2FA al móvil y el resultado al usuario.
- **App 2FA:** autenticador móvil; muestra la petición y envía "aprobar" o "rechazar" al backend.
- **D2OTP Lib Kotlin:** librería cliente que ayuda a cifrar/descifrar (generar y leer) los mensajes OTP en las apps móviles.
- **D2OTP Lib Python:** librería servidor que ayuda a cifrar/descifrar (generar y leer) los mensajes OTP en el backend.

4.3 Vista de despliegue e infraestructura

4.3.1 Entorno de ejecución

El sistema se despliega, en su implementación actual, sobre **tres dominios de ejecución** claramente diferenciados:

Dominio	Elementos hospedados	Detalles de red / puertos	Seguridad aplicada
Dispositivo móvil (Android 11 +)	• App Cliente (banca). • App 2FA. • Librería D2OTP (Kotlin).	Salida HTTPS 443 → Backend. FCM *outbound* 5228–5230 / 443 para notificaciones push [26].	Android Keystore para K_{AB}; **TLS 1.3**; verificación de firma de la APK.
Servidor local de desarrollo	• Backend FastAPI [23] + Uvicorn (Docker). • Módulo D2OTP (Python/Cython).	localhost:8000 (interno) → expuesto por **ngrok** a URL https://<rand>.ngrok-free.app sobre 443[22].	Certificado TLS gestionado por ngrok; autenticación vía token de agente; log IP restringido.
Google Cloud	• Cloud Firestore (multi-region). • Firebase Cloud Messaging (FCM). • Opcional: SMS Gateway (Cloud Functions / Twilio).	Firestore gRPC/HTTPS 443[27]. FCM 5228–5230/443[28].	Datos cifrados en reposo con claves Google o CMEK [24]; reglas de acceso específicas [25].

Flujo básico de creación de una petición de login 2FA

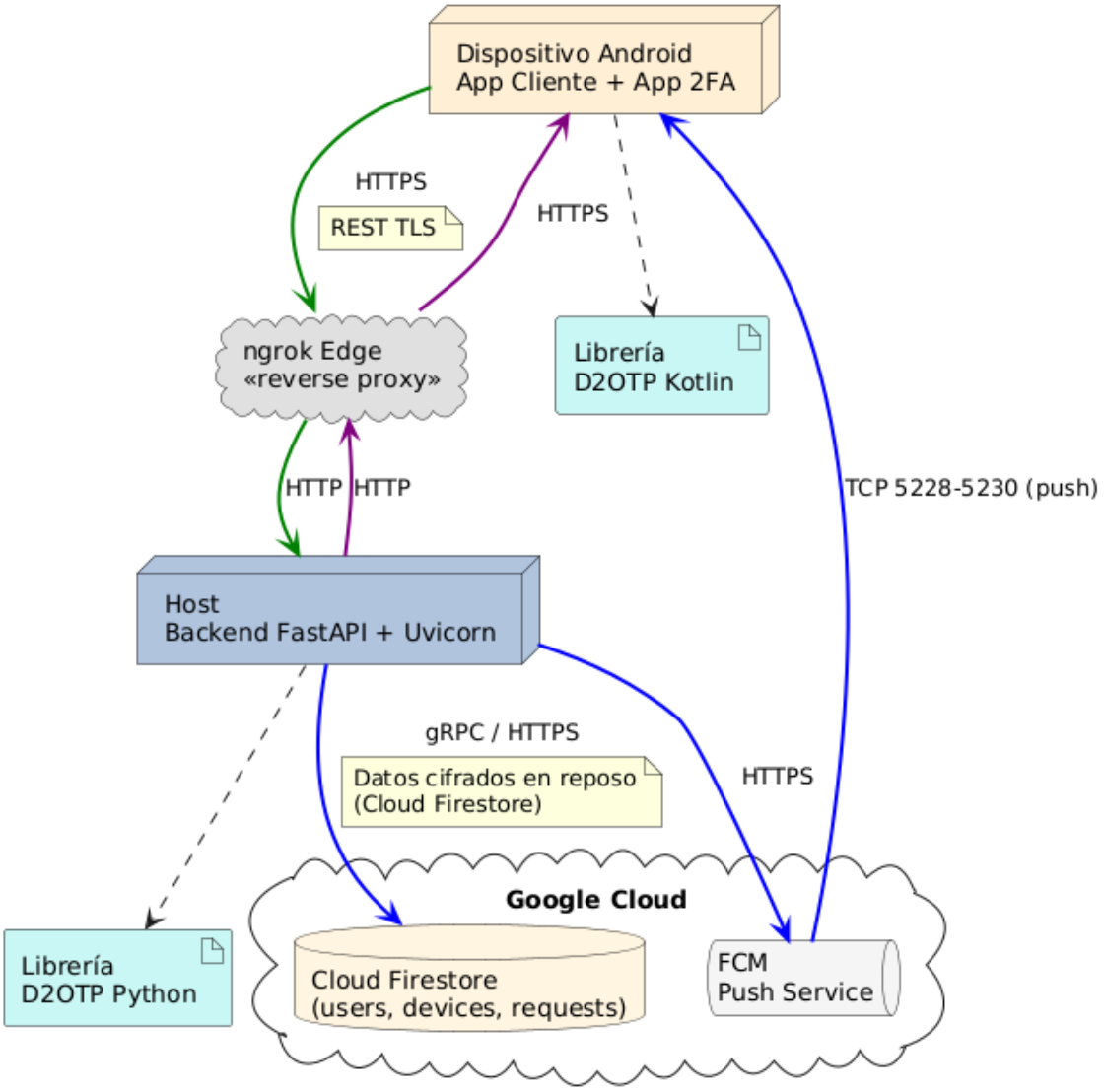
1. Tras un login con las credenciales correctas, la App Cliente invoca POST /crear_solicitud en el backend junto con un mensaje cifrado M1 que contiene los datos de la solicitud de inicio de sesión que se va a generar (túnel ngrok).

2. El backend registra la solicitud en Firestore y envía un **M1** (distinto al de la app cliente) que contiene los datos de la solicitud a la App 2FA vía FCM; fallback SMS opcional.

3. La App 2FA decodifica el M1 que ha recibido, solicita interacción al usuario e invoca /respuestaPush con un **M2** que contiene la respuesta a la petición (aceptada/denegada) y una nueva clave temporal CCAB para futuras comunicaciones entre backend y App 2FA (REST o FCM upstream).

4. El backend valida **M2**, actualiza CCAB en Firestore, actualiza la petición en Firestore y notifica a la App Cliente con un **M2** que contiene la respuesta de App 2FA ante la solicitud (aceptada/denegada) y una nueva clave CCAB para futuras comunicaciones entre App cliente y backend, habiendo sido actualizada dicha clave en el Firestore previo al envío.

4.3.2 Diagrama de despliegue (UML)

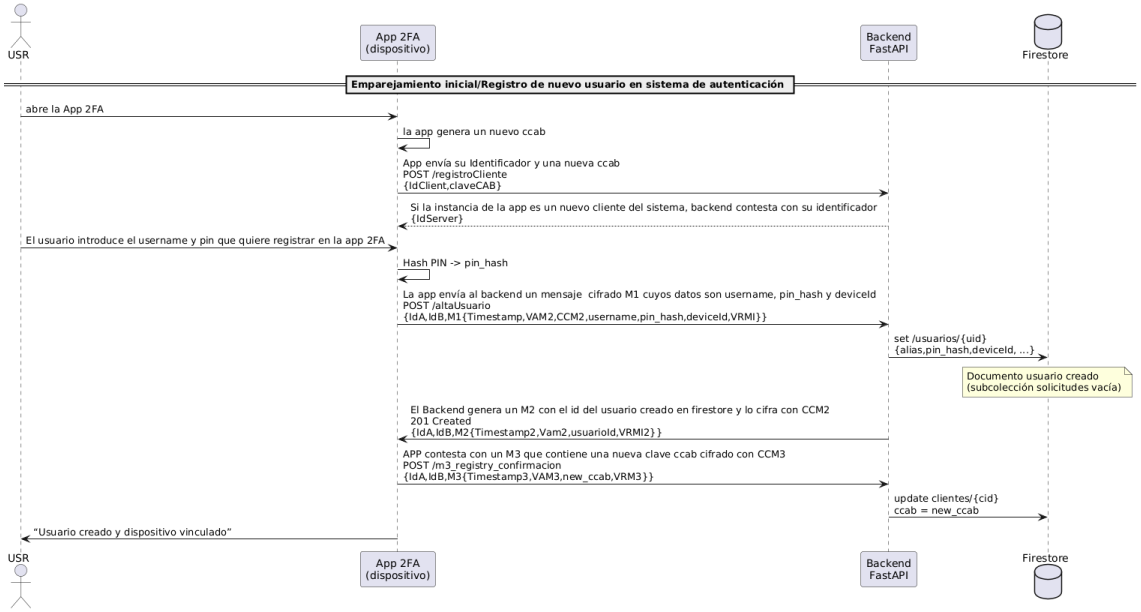


El diagrama ilustra cómo todos los distintos elementos del sistema están interconectados entre sí.

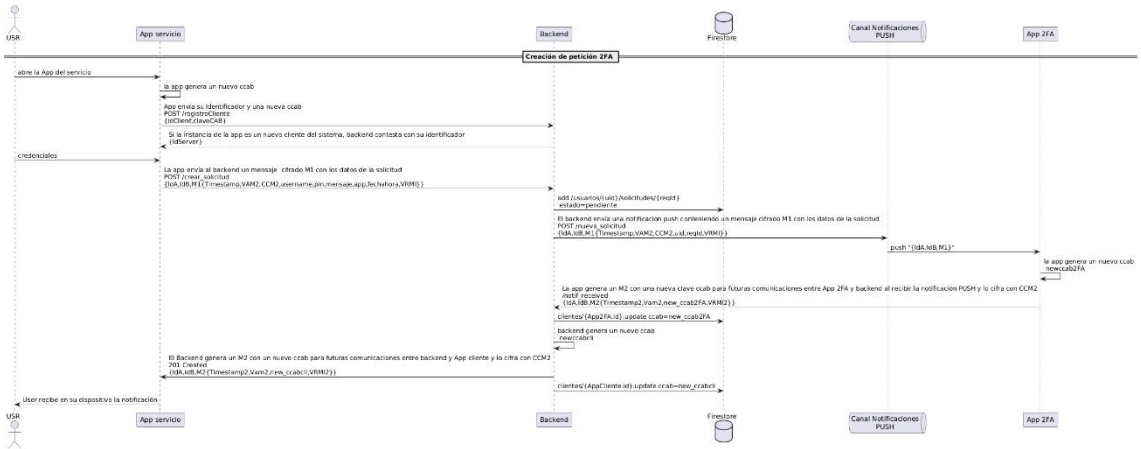
Es importante hacer incapié en que **todas las rutas de datos sensibles quedan encapsuladas** en TLS y cifrado D2OTP; la única exposición pública es el subdominio ngrok, que actúa como *reverse proxy* sobre el backend local. Para producción, basta con reemplazar ngrok por un *load-balancer* cloud manteniendo las mismas interfaces, cumpliendo así con el objetivo **O-3 Escalabilidad horizontal** definido en 4.1.

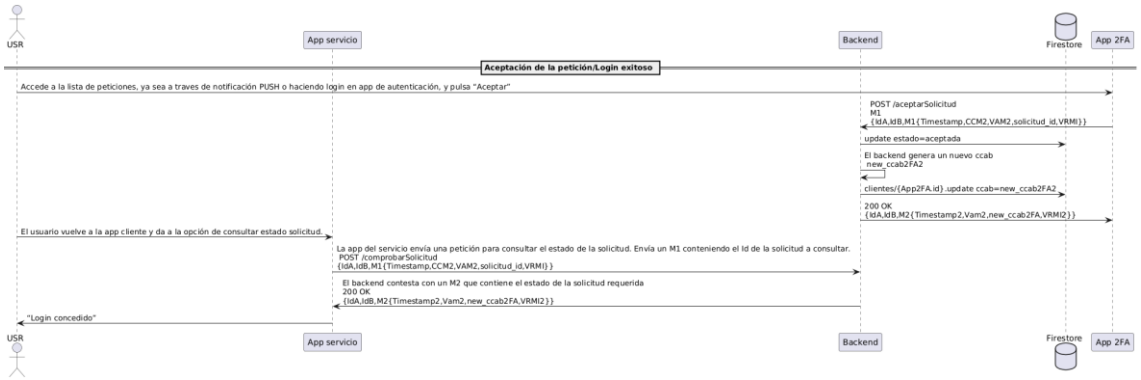
4.4 Diagramas de secuencia

4.4.1 Registro de dispositivo/usuario

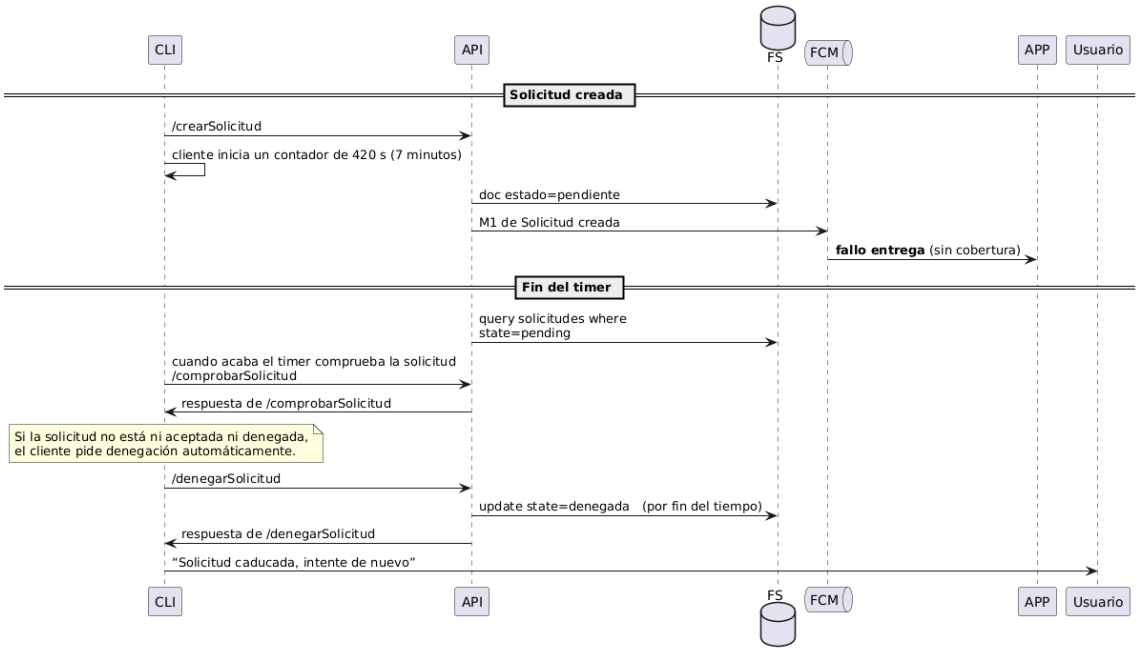


4.4.2 Autenticación M1 – M2 – M3





4.4.3 Manejo de errores / expiración



4.5 Consideraciones de seguridad en el diseño

4.5.1 Cifrado extremo a extremo (D2OTP)

Se establece como requisito fundamental que **todas las comunicaciones** (a excepción de la inicial, en la que dos componentes comparten su primera clave CCAB) entre los distintos componentes del sistema estén **cifradas de extremo a extremo**, conforme al protocolo **D2OTP**. Este enfoque garantiza la confidencialidad e integridad de los datos transmitidos, evitando accesos no autorizados o manipulaciones durante el tránsito.

4.5.2 Reglas de acceso Firestore

Para maximizar la seguridad en el acceso a la base de datos, se ha decidido que **únicamente el backend** debe tener permisos de lectura y escritura sobre Firestore. De este modo, se evita que los clientes (como las aplicaciones móviles) interactúen directamente con la base de datos, reduciendo así el riesgo de exposición o manipulación indebida de la información.

Las reglas de seguridad implementadas en Firestore son las siguientes:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth.uid=backendId;
    }
  }
}
```

Esta configuración asegura que sólo las solicitudes autenticadas provenientes del backend (identificado mediante backendId) puedan acceder o modificar los documentos de Firestore.

4.5.3 Protecciones en la app móvil

Además del cifrado extremo a extremo mencionado previamente, la aplicación móvil implementa una verificación adicional basada en el **identificador del dispositivo (deviceId)**. Antes de permitir el acceso a los datos del usuario, se comprueba que el identificador del dispositivo desde el cual se realiza la petición coincida con el registrado para dicho usuario.

En caso de que no coincidan, se genera una **solicitud de cambio de dispositivo vinculado**, la cual **debe ser aprobada desde el dispositivo actualmente autorizado**. Esta medida aporta una capa adicional de

seguridad frente a accesos no autorizados, aunque implica el riesgo potencial de **pérdida de acceso a la cuenta** si el usuario pierde su dispositivo.

Para mitigar este riesgo, se recomienda implementar un **sistema de claves de recuperación** durante el proceso de registro inicial del usuario. Esta funcionalidad no ha sido incluida en la implementación desarrollada debido a limitaciones de tiempo.

4.6 Decisiones arquitectónicas clave

4.6.1 Justificación de FastAPI + KMM + Firebase

La arquitectura del sistema ha sido diseñada con criterios de seguridad, eficiencia, escalabilidad, y sobre todo facilidad de desarrollo, alineados con los requerimientos del protocolo D2OTP. En este contexto, se seleccionaron tres componentes principales: **FastAPI** para el backend, **Kotlin Multiplatform Mobile (KMM)** para la aplicación móvil, y **Firebase** como infraestructura escalable. A continuación, se justifica técnicamente esta elección.

FastAPI para el Backend

Se optó por FastAPI como framework para implementar la API RESTful central del sistema, dada su capacidad para ofrecer desarrollo ágil, validación automática de datos y generación de documentación interactiva (Swagger UI). Esto facilita tanto la implementación como la presentación académica del proyecto. Asimismo, al tratarse de un backend ejecutado de forma local o en servidor propio, permite un control completo sobre la seguridad, cifrado y gestión de sesiones, aspectos fundamentales en un sistema de autenticación fuerte [19].

Kotlin Multiplatform Mobile (KMM) para la App Móvil

La elección de KMM responde a su capacidad para compartir lógica entre plataformas sin sacrificar el acceso nativo a funcionalidades críticas del sistema operativo. En concreto, KMM permite la gestión segura de claves a través del Android Keystore, la ejecución eficiente de tareas criptográficas, y la integración directa con servicios backend como Firebase Cloud Messaging (FCM). Estos factores fueron determinantes, dada la naturaleza del sistema, que requiere autenticación mutua, manejo de OTP dinámicos y cifrado de alto nivel [29].

Firebase como Plataforma de Infraestructura

Firebase fue seleccionado como solución cloud principal debido a su integración optimizada con Android/Kotlin, su soporte nativo para notificaciones PUSH (a través de FCM), almacenamiento seguro (Firestore y Firebase Storage) y escalabilidad automática. Adicionalmente, su compatibilidad multiplataforma lo posiciona como una opción ideal para futuras expansiones a iOS o Web [30].

4.6.2 Trade-offs y alternativas descartadas

Durante el proceso de diseño e implementación del sistema, se evaluaron múltiples alternativas tecnológicas para los tres componentes principales: backend, aplicación móvil e infraestructura. Esta sección expone los **trade-offs** observados y aceptados con las tecnologías elegidas (FastAPI, KMM y Firebase), así como un análisis de las alternativas descartadas.

FastAPI (Backend)

Trade-offs aceptados:
FastAPI ofrece una sintaxis clara, tipado fuerte y validación automática, lo que acelera el desarrollo de APIs RESTful. Sin embargo, al tratarse de un framework relativamente reciente, su ecosistema y comunidad son más limitados comparado con opciones más consolidadas como Flask o Django. Además, aunque permite despliegue local, no proporciona por sí mismo tolerancia a fallos ni escalabilidad automática, lo que implicaría infraestructura adicional para escenarios de alta disponibilidad.
Debido al alcance limitado de este Trabajo de Fin de Grado, no se ha considerado que esas cuestiones pudiesen llegar a ser un problema

Alternativas descartadas:

- **Flask**: menos estructurado y con mayor necesidad de configuración manual.
- **Node.js (Express)**: mayor complejidad en la gestión de seguridad criptográfica.
- **AWS Lambda / Google Cloud Functions**: poco adecuadas para manejar claves persistentes o mantener estados críticos.
- **Arquitectura de microservicios**: demasiado compleja para el alcance de este TFG.

Kotlin Multiplatform Mobile (KMM)

Trade-offs aceptados:
KMM permite compartir lógica de negocio entre plataformas y ofrece acceso directo a las APIs nativas de Android, siendo apto para proyectos con altos requerimientos de seguridad como este. No obstante, aún se encuentra en fase experimental (en el momento del desarrollo del proyecto), lo que implica cierta inestabilidad y menor soporte en herramientas de terceros, especialmente para la parte iOS. Además, su curva de aprendizaje es más pronunciada comparada con frameworks como Flutter o React Native. En cualquier caso, debido al hecho de que este TFG no se ha desarrollado con iOS en mente por falta de tiempo y hardware, estos trade-offs no se consideran un problema

Alternativas descartadas:

- **Flutter**: Excelente en experiencia visual y productividad, pero limitado en acceso a funcionalidades nativas avanzadas de seguridad, y dependiente de Dart, un lenguaje menos adoptado en entornos críticos.

- **React Native**: Popular y bien documentado, pero presenta restricciones significativas en rendimiento criptográfico, almacenamiento seguro y tareas en segundo plano, todos ellos aspectos críticos en la implementación del protocolo D2OTP.

Firebase (Infraestructura cloud)

Trade-offs aceptados:
Firebase ofrece integración directa con Kotlin y Android, notificaciones push, almacenamiento seguro (Firestore, Storage), y escalabilidad automática con bajo coste inicial. Sin embargo, esta elección implica una fuerte dependencia del ecosistema Google, lo cual podría limitar la portabilidad o el control a nivel bajo de ciertos procesos (por ejemplo, cifrado en bases de datos o despliegues personalizados). Además, para ciertos casos de uso empresariales o regulatorios, Firebase puede no ofrecer el nivel de granularidad requerido. Debido al alcance limitado de este TFG, no se ha considerado cambiar a otra infraestructura. Además, debido al carácter modular del sistema desarrollado, reemplazar firebase por cualquier otra infraestructura es sencillo.

Alternativas descartadas:

- **Amazon Web Services (AWS)**: curva de aprendizaje más pronunciada y mayor complejidad de configuración.
- **Microsoft Azure**: integración menos fluida en el entorno móvil Android y menor popularidad en proyectos académicos pequeños.

4.7 Resumen de diseño/arquitectura

En este capítulo se ha descrito en detalle el diseño arquitectónico del sistema de autenticación multifactor basado en el protocolo D2OTP. El diseño se ha estructurado en torno a objetivos fundamentales como la seguridad extremo a extremo, la modularidad, la escalabilidad horizontal y la compatibilidad multiplataforma. A partir de estos principios, se definieron los requisitos funcionales y no funcionales que guiaron la selección de tecnologías y la organización de los componentes.

Se ha optado por una arquitectura distribuida cliente-servidor, con un backend RESTful desarrollado en FastAPI, una aplicación móvil implementada mediante Kotlin Multiplatform Mobile (KMM), y una infraestructura en la nube basada en Firebase para mensajería, almacenamiento y escalabilidad. Esta combinación permite garantizar comunicaciones seguras, un despliegue ágil, y una integración futura con otras plataformas como iOS o Web.

Los diagramas de componentes, despliegue y flujo de datos han permitido ilustrar el funcionamiento operativo del sistema, así como su comportamiento en escenarios reales como el emparejamiento de dispositivos, el proceso M1–M3 de autenticación, y el manejo de errores y expiraciones.

En cuanto a la seguridad, se han incorporado medidas como cifrado extremo a extremo mediante D2OTP, reglas estrictas de acceso a Firestore, y verificación del deviceId en el cliente móvil. Se identificaron limitaciones no críticas, como la falta de implementación de notificaciones push o de claves de recuperación, debido a restricciones de tiempo, aunque su impacto funcional es reducido.

Por último, se justificó la elección de las tecnologías principales, analizando sus ventajas técnicas, compromisos aceptados (*trade-offs*), y comparándolas con alternativas descartadas. Esta evaluación crítica respalda la coherencia del diseño adoptado, tanto desde el punto de vista técnico como académico.

5.Implementación

En esta sección queda detallada mi implementación del sistema

Parte	Lenguaje / SDK	Entorno de ejecución
Librería D2OTP	Kotlin Multiplatform Module 2.0 (parte Android) + Python 3.11 (parte Backend)	Android Studio/Uvicorn
Backend	Python 3.11 + FastAPI 0.111[19]	Uvicorn
App Banco	Kotlin Multiplatform 2.0[29]	Android Studio
App 2FA	Kotlin Multiplatform 2.0[29]	Android Studio

Nota:

Si bien en secciones anteriores esta memoria se detalla de forma clara cómo debe realizarse la integración de **notificaciones push**, dicha funcionalidad **no ha sido implementada en esta versión del proyecto** debido a limitaciones de tiempo durante el desarrollo.

No obstante, esta ausencia **no compromete la funcionalidad general del sistema**, ya que se proporciona el usuario puede iniciar sesión en la aplicación de autenticación. De este modo, las operaciones pueden llevarse a cabo correctamente sin depender de las notificaciones push.

Respecto a las comunicaciones mediante SMS, que tampoco están implementadas, los motivos son los mismos, limitaciones de tiempo.

Considero más relevante implementar primero la integración de notificaciones push antes de investigar sobre SMS.

5.1 Librería D2OTP

La librería D2OTP se compone de dos implementaciones diferentes, una librería en Python exclusivamente para el backend y otra escrita en Kotlin tanto para aplicaciones cliente android que quieran integrar el sistema como para la propia aplicación de verificación 2FA.

5.1.1 Funciones librería parte android

- **sha256(input: String): String**

Genera un hash SHA-256 de input y lo codifica en base64.

- **generateRandomValue(length: Int = 16): String**

Genera un valor aleatorio seguro de length bytes (por defecto 16), codificado en base64. Se usa para claves y valores de verificación.

- **aesEncrypt(plainText: String, base64Key: String): String**

Cifra un texto plano con **AES en modo ECB y padding PKCS5**, usando una clave codificada en base64. Devuelve el **resultado en base64**.

- **aesDecrypt(encryptedBase64: String, base64Key: String): String**

Descifra un texto (base64) con **AES ECB y PKCS5**, usando la misma clave en base64. **Devuelve el texto plano original**.

- **generateVRMI3M(idA: String, idB: String, timestamp: String, CCM2: String, VAM2: String, CCM3: String, VAM3: String, datosI: String): String**

Genera el **hash VRM** correspondiente a un mensaje **M1** que será **seguido de un M2 y un M3** a lo largo de la comunicación.

- **generateVRMI2M(idA: String, idB: String, timestamp: String, CCM2: String, VAM2: String, datosI: String): String**

Genera el **hash VRM** correspondiente a un mensaje **M1** que será **seguido de sólo un mensaje M2** a lo largo de la comunicación.

- **generateVRMI1M(idA: String, idB: String, timestamp: String, datosI: String): String**

Genera el **hash VRMI** correspondiente a un mensaje **M1** que **no será respondido** con ningún mensaje M2.

- **generateVRM2(idA: String, idB: String, timestamp: String, CCM2: String, VAM2: String, datosI: String): String**

Genera el **hash VRM** correspondiente a un mensaje **M2**.

- **generateVRM3(idA: String, idB: String, timestamp: String, CCM3: String, VAM3: String, datosI: String): String**

Genera el **hash VRM** correspondiente a un mensaje **M3**.

5.1.2 Funciones librería parte backend

- **generate_random_value(bytes_length=16) -> str**

Genera un valor aleatorio de bytes_length (por defecto 16) y lo codifica en base64. Se usa para claves principalmente y valores de verificación.

- **aesEncrypt(plainText: String, base64Key: String): String**

Cifra un texto con AES en **modo ECB con relleno PKCS#7**, usando una clave codificada en base64. Devuelve el resultado en base64.

- **aesDecrypt(encryptedBase64: String, base64Key: String): String**

Descifra un texto cifrado en base64 usando **AES-ECB y elimina el padding** para recuperar el texto original.

5.2 Backend FastAPI

El backend está contenido dentro de una carpeta con los siguientes elementos:

- **xxxxx-firebase-adminsdk-fbsvc-xxxxxx.json** – Credenciales de servicio de Firebase (archivo JSON).
- **ComoIniciarAPI.txt** – Documento de texto con instrucciones para poner en marcha la API.
- **database.py** – Módulo que gestiona la conexión a Firestore. Es muy importante indicarle el nombre exacto del json de la firebase que queramos utilizar.
- **libreriaD2OTPPython.py** – Parte Backend de la librería D2OTP.
- **main.py** – Punto de entrada de la aplicación FastAPI (crea la instancia de FastAPI).
- **routes.py** – Archivo donde se declaran los endpoints (router) de la API.

Para ejecutarlo hay que situarse en el directorio raíz del backend y lanzarlo con:

```
uvicorn main:app --host 0.0.0.0 --port 8000
```

Para exponer el servicio se empleó un túnel **ngrok** (plan gratuito). Al reiniciar ngrok la URL cambia; por ello es **necesario actualizar manualmente** el valor **baseUrl** en ApiClient.kt antes de cada arranque de las aplicaciones móviles.

Todos los endpoints, con la excepción de /registroCliente, reciben información cifrada siguiendo el protocolo D2OTP, la descifran, realizan la acción pertinente, y generan una respuesta correctamente cifrada bajo dicho protocolo y la envían a quién invocó el endpoint en cuestión.

5.2.1 Cuerpos de solicitud

Estas son los 3 tipos de clase que pueden recibir los endpoints

- **Solicitudes de registro de nuevo cliente:**

```
class RegisterClientRequest(BaseModel):
    idClient: str
```

```
    claveCAB: str
```

- **Mensajes M1:**

```
class M1Request(BaseModel):
    idA: str
    idB: str
    M1: str
```

- **Mensajes M3:**

```
class M3Confirmacion(BaseModel):
    idA: str
    idB: str
    M3: str
```

5.2.2 Endpoints

POST /registroCliente
Es la única comunicación que se envía sin cifrar siguiendo el protocolo D2OTP.

Se envía cada vez que se inicia una aplicación cliente o la propia App 2FA, este endpoint recibe un **identificador de la aplicación** (idClient) y una **clave temporal**(ccab) que será usada para cifrar y descifrar los M1.El backend busca en la lista de clientes del firebase si hay algún cliente existente cuyo IdA es el idClient dado, **si ya existe lanza una excepción**, y si no existe da de alta a un cliente A creando una entrada en la lista de clientes con idClient en su IdA y ccab en su elemento ccab. Devuelve un identificador fijo del servidor (id_backend) (esto es así porque los id, ya sean de app cliente, app 2FA, o del propio backend, nunca cambian). Se invoca siempre que se inician las aplicaciones, pero sólo funciona y retorna id_backend la primera vez que se inicia una intalación de una de las aplicaciones cliente/2FA.

POST /altaUsuario
Recibe el **M1 de registro** que la app 2FA genera para crear un nuevo usuario del sistema y que contiene un nombre de usuario, el hash de una contraseña/pin, y el deviceId del dispositivo desde el que se creó el usuario. Tras verificar firma e integridad correctamente, crea el documento usuarios/{uid} con esos 3 elementos y un cuarto elemento creado por el backend fecha_creacion que indica la fecha de creación, y responde con un mensaje **M2 cifrado** que contiene el identificador del documento creado.

POST /m3_altaUsuario_confirmacion
La App 2FA envía aquí el **M3** que cierra el registro, que contiene una nueva contraseña ccab. El backend descifra M3, valida el hash VRM3 y el timestamp

(que debe ser anterior al tiempo actual) y cambia el CCAB asignado a la app 2FA, dejando los datos de la app 2FA correctamente actualizados para futuras comunicaciones cifradas.

POST /login_d2otp
Maneja el **M1 de inicio de sesión**, que contiene un nombre de usuario y el hash de un pin/contraseña. Descifra el mensaje, valida VRMI (hash del contenido de M1) y genera un M2 (incluye uid y deviceId) con el que contesta a la app que lo invocó (app cliente o app 2FA.

POST /m3_login_confirmacion
Punto de llegada del **M3 de login** conteniendo una nueva CCAB generada por la app que invocó al login. Al igual que el endpoint **/m3_altaUsuario_confirmacion**, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

POST /crear_solicitud
Procesa el **M1 de creación de solicitud** (p. ej., inicio de sesión). Crea el documento solicitudes/{reqId} dentro usuarios/{userId}, en dicho documento introduce los datos correspondientes contenidos en el M1 (aplicación que envía la solicitud, fecha y hora de solicitud, mensaje que indica el tipo de solicitud, e id de la solicitud), pone el estado (otro campo dentro de la solicitud) como pendiente, y responde con un M2 cifrado que contiene el id del usuario que creó la solicitud y el id de la propia solicitud.

POST /crear_solicitud_confirmacion
Punto de llegada del **M3 de creación de solicitud** conteniendo una nueva CCAB generada por la app que invocó al login. Al igual que el endpoint **/m3_altaUsuario_confirmacion**, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

POST /solicitudesPendientes
La App 2FA envía un M1 que contiene el usuarioId cuya lista de solicitudes queremos consultar, descifra y verifica M2, recorre usuarios/{usuarioId}/solicitudes en la firebase mirando todas las solicitudes cuyo estado sea pendiente, las mete en una lista y retorna a la app 2FA dicha lista contenida dentro de un mensaje M2 cifrado.

POST /m3_solicitudesPendientes_confirmacion
Complementa el punto anterior, obtiene un **M3 de solicitudes pendientes** conteniendo una nueva CCAB generada por la app que invocó a /solicitudesPendientes. Al igual que el endpoint **/m3_altaUsuario_confirmacion**, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

POST /aceptarSolicitud
Recibe el M1 con usuario y solicitud_id, descifra y verifica dicho M1 y cambia el estado de la solicitud contenida en usuarios/{usuario_id}/solicitudes/{solicitud_id} a **“aceptada”**. Luego genera

nuevo CCAB, lo cambia en la página del cliente que mandó la solicitud y devuelve un M2 con el nuevo CCAB contenido dentro.

POST /denegarSolicitud
Recibe, hace y devuelve exactamente lo mismo que **/aceptarSolicitud**, pero cambia el estado a **"denegada"** (y obviamente también cambia el CCAB por el nuevo).

POST /obtenerSolicitud
Recibe un M1 que contiene el id de una solicitud que se desea consultar, descifra y verifica M1, busca y obtiene todos los datos de la solicitud cuyo id sea el contenido en M1, actualiza la ccab del cliente y devuelve un M2 cifrado que contiene dicha solicitud de una solicitud concreta indicada en el M1 y el nuevo CCAB.

POST /comprobarSolicitud
Recibe un M1 que contiene el id de una solicitud que se desea consultar, descifra y verifica M1, busca y obtiene el estado de la solicitud queremos consultar y devuelve un M2 cifrado que contiene el estado de dicha solicitud.

POST /m3_comprobarSolicitud_confirmacion
Complementa el punto anterior, obtiene un **M3 de comprobar solicitud** conteniendo una nueva CCAB generada por la app que invocó a /comprobarSolicitud. Al igual que el endpoint **/m3_altaUsuario_confirmacion**, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

POST /vincularDispositivo
Recibe un M1 cifrado que contiene los datos de una solicitud de vinculación y lo descifra y verifica, después, al igual que /crear_solicitud, crea una nueva solicitud, esta vez añadiendo un campo extra que contiene el deviceId del dispositivo que quiere reemplazar al actual. Finalmente crea y cifra un nuevo M2 que contiene el userId del usuario en el que se ha creado la solicitud y la solicitudId de la solicitud creada.

POST /m3_vincularDispositivo_confirmacion
Complementa el punto anterior, obtiene un **M3 de vincular dispsitivo** conteniendo una nueva CCAB generada por la app que invocó a /vincularDispositivo. Al igual que el endpoint **/m3_altaUsuario_confirmacion**, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

POST /cambiarDevice
Recibe un M1 cifrado que contiene un ud de usuario y un id de solicitud, lo descifra y verifica, y después, busca la solicitud en cuestión. Una vez encontrada, obtiene el dato contenido en su campo deviceId, que corresponde al deviceId del nuevo dispositivo, y se guarda en el campo device_id del usuario cuyo identificador es userId. Finalmente crea, cifra y envía un nuevo M2 que contiene el nuevo device Id.

POST /m3_cambiarDevice_confirmacion
Complementa el punto anterior, obtiene un **M3 de cambiar dispositivo**

conteniendo una nueva CCAB generada por la app que invocó a /cambiarDevice. Al igual que el endpoint /m3_altaUsuario_confirmacion, si el VRM3 y el timestamp son correctos, cambia la CCAB por la contenida en el M3 y confirma que el inicio de sesión fue válido.

5.3 Aplicación bancaria mock (ejemplo de app cliente)

5.3.1 Pantallas

- **FirstScreen.kt**

Pantalla de arranque (*splash*). Muestra el logotipo de la entidad y un mensaje breve; tras unos segundos redirige automáticamente a WelcomeScreen.

- **WelcomeScreen.kt**

Pantalla de inicio con un botón **Login** que abre LoginScreen.

- **LoginScreen.kt**

Formulario de autenticación (alias + PIN). Si el login es correcto, se lanza una solicitud de inicio de sesión 2FA, y se va a WaitingScreen mientras la solicitud es atendida desde App 2FA

- **WaitingScreen.kt**

Pantalla intermedia que muestra un contador regresivo mientras espera la respuesta 2FA del dispositivo vinculado.

- **Aceptado** → AcceptedScreen.
- **Denegado** → DeniedScreen.
- **Timeout** (no hubo respuesta) → FailedLoginScreen.
- **AcceptedScreen.kt**

Mensaje de éxito: “"Has aceptado la petición con éxito”. Si estuviese correctamente implementado, tendría un botón **Continuar** que redirigiría al panel principal de la aplicación bancaria, sin embargo, actualmente tiene un botón **Volver al inicio** que retorna a WelcomeScreen.

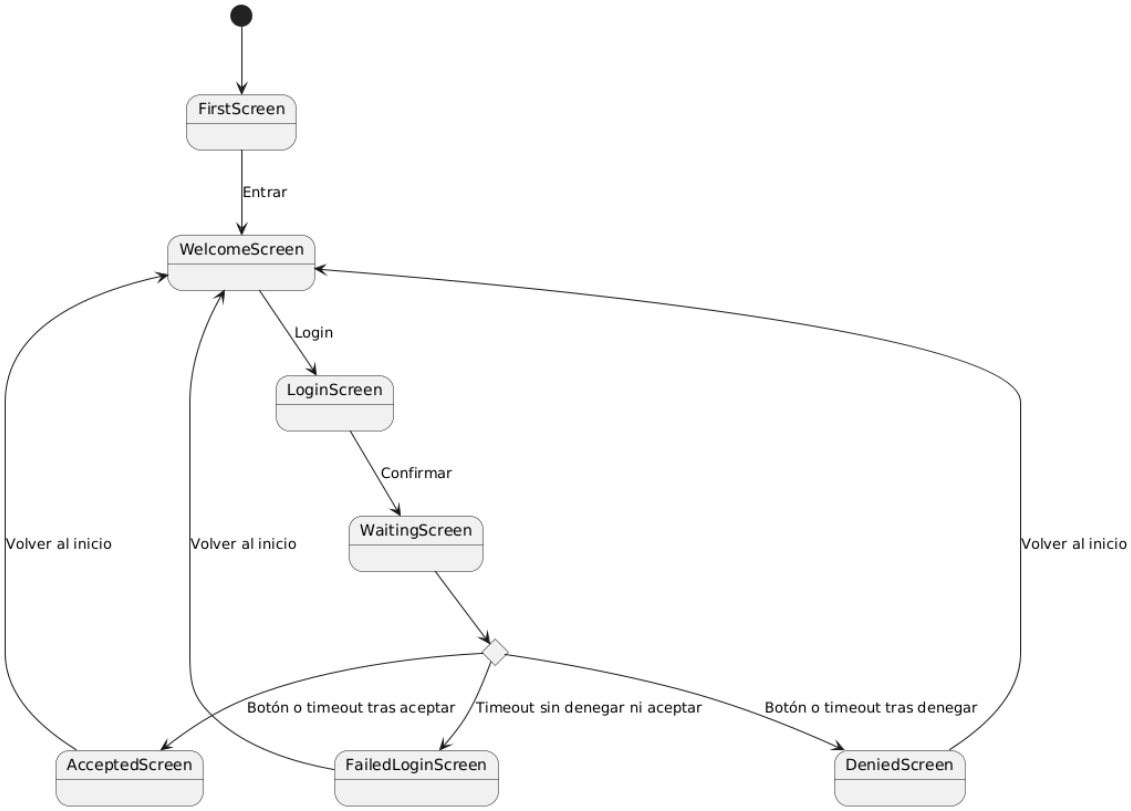
- **DeniedScreen.kt**

Muestra “Solicitud denegada” e indica que la operación fue rechazada desde el dispositivo 2FA. Botón **Volver al inicio** para regresar a WelcomeScreen.

- **FailedLoginScreen.kt**

Indica fallo de autenticación por timeout. Botón **Volver al inicio** para volver a WelcomeScreen.

5.3.2 Diagrama de estados UML



5.3.3 Flujo típico desde apertura hasta inicio de sesión exitoso

1. El usuario abre la aplicación y pulsa **Entrar** (FirstScreen → WelcomeScreen).
2. Pulsa **Iniciar Sesión** para entrar a LoginScreen (WelcomeScreen->LoginScreen).
3. Para autenticarse, introduce alias y PIN. Se envía una solicitud de inicio de sesión y se inicia un contador de 420 segundos para poder autenticarla (LoginScreen->WaitingScreen).
4. El usuario abre la app 2FA, inicia sesión y acepta la solicitud allí. Después, volviendo a la app cliente, pulsa el botón de “Ya he aceptado

la solicitud" o espera a que se acabe el tiempo, y accede finalmente a su cuenta (WaitingScreen->AcceptedScreen).

5.4 Aplicación de autenticación

5.4.1 Pantallas

- **FirstScreen.kt**

Pantalla de arranque. Muestra el texto "App D2OTP" y un botón "entrar", que al ser pulsado envía POST a /registroCliente y redirige a WelcomeScreen.

- **WelcomeScreen.kt**

Menú inicial con dos opciones:
-"Crear cuenta nueva" → abre RegisterScreen.
-"Login" → abre LoginScreen.

- **RegisterScreen.kt**

Formulario de alta de usuario (alias + PIN). Envía POST a /altaUsuario. Al completarse con éxito, navega a SuccessfulRegistryScreen.

- **SuccessfulRegistryScreen.kt**

Mensaje "Éxito al crear la cuenta" y botón para volver al menú principal (WelcomeScreen).

- **LoginScreen.kt**

Entrada de alias y PIN. Envía el M1 de /login_d2otp y, si la autenticación es correcta, y desde el dispositivo vinculado con el usuario introducido, pasa a AuthRequestsScreen. En caso de que la autenticación sea correcta peor el dispositivo no sea el vinculado con la cuenta introducida, pasa a UnlinkedDeviceScreen.

- **AuthRequestsScreen.kt**

Lista de solicitudes 2FA pendientes enviadas por el backend (/solicitudesPendientes). Al tocar un elemento abre RequestDetailScreen.

- **RequestDetailScreen.kt**

Detalle de la solicitud (importe, destino, etc.) con botones **Aceptar** y **Denegar**. Tras la acción vuelve a WelcomeScreen. Esto es así porque si se volviese a AuthRequestScreen tras aceptar solicitudes de cambio de dispositivo vinculado, el dispositivo viejo desvinculado seguiría con la sesión iniciada y pudiendo gestionar más solicitudes.

- **UnlinkedDeviceScreen.kt**

Se muestra cuando se ha intentado iniciar sesión en un usuario para el cual el dispositivo no está vinculado, y manda automáticamente una petición de dispositivo para ser gestionada desde el dispositivo vinculado. Si se pulsa el

botón después de atender la solicitud o se acaban los 420 segundos del timer, se llega a SuccessfullLinkScreen o FailedLinkScreen (dependiendo de si se aceptó, denegó o ignoró la solicitud).

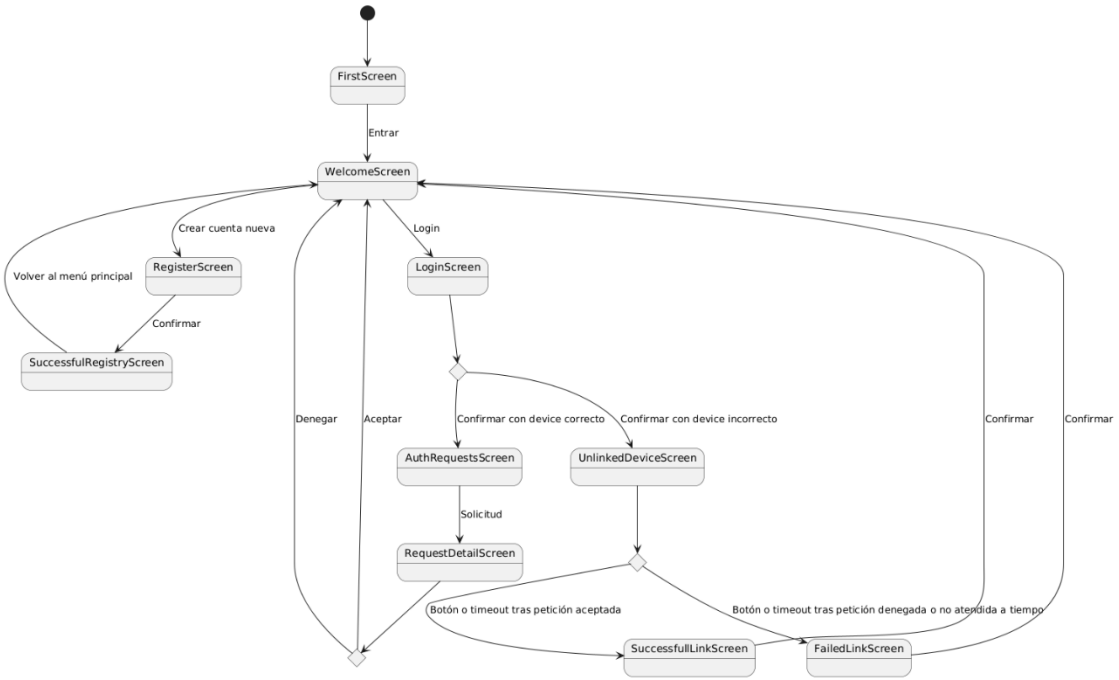
- **SuccessfulLinkScreen.kt**

Pantalla de confirmación cuando la vinculación de un nuevo dispositivo se realiza con éxito. Botón “Confirmar” para regresar a WelcomeScreen.

- **FailedLinkScreen.kt**

Informa de que se denegó o no se atendió a tiempo la solicitud de vinculación de dispositivo. Botón “Confirmar” para regresar a WelcomeScreen.

5.4.2. Diagrama de estados UML



5.4.3 Flujo típico desde apertura hasta aprobación de solicitud

5. El usuario abre la aplicación y pulsa **Entrar** (FirstScreen → WelcomeScreen).
6. Si es un nuevo usuario, completa el registro (RegisterScreen → SuccessfulRegistryScreen->WelcomeScreen).
7. Pulsa Iniciar sesión y, para autenticarse, introduce alias y PIN (WelcomeScreen->LoginScreen).

8. El backend valida las credenciales y se hizo login desde el dispositivo correcto (LoginScreen->AuthRequestsScreen) o muestra **UnlinkedDeviceScreen** si no coincide.

9. En AuthRequestsScreen, cada solicitud pendiente se muestra en una lista para poder ser aprobada/denegada desde **RequestDetailScreen**, tras lo cual se retorna a la pantalla de bienvenida.

5.5 Modelo de datos en Firebase (Firestore)

Ruta / Colección	**Documento -> Campos principales**	**Comentarios**
usuarios	**Documento UID** -> • `alias : String` • `device_id : String` *(UUID móvil)* • `pin_hash : String` *(Argon2id)* • `fecha_creacion : Timestamp`	Cada usuario almacena **un único dispositivo activo.**
usuarios/{uid}/solicitudes	**Documento reqId** -> • `id : String` *(reqId)* • `app : String` *(ej. "Fakebank")* • `mensaje : String` *(contexto)* • `estado : String` • `fecha : String`	Sub-colección **anidada** dentro de cada usuario. No existe colección global de solicitudes.
clientes	**Documento clild ->** • `idA : String` • `ccab : String` • `fecha_registro : String`	Se debe **actualizar CCAB** tras cada comunicación entre una parte y el backend.

5.6 Limitaciones de la implementación conocidas

- **Despliegue de laboratorio**: backend tras túnel **ngrok** con Url en constante cambio. En las aplicaciones, se necesita editar el valor de la Url de la API en cada despliegue del mismo.
- **Solo Android**: la rama iOS de las aplicaciones KMM no se ha desarrollado por falta de hardware y tiempo.
- **Gestión de dispositivos limitada**: actualmente el sistema sólo admite tener asignado un deviceId a cada usuario, y hay que aceptar una petición en el antiguo dispositivo para vincular el nuevo.

6. Pruebas y validación

A fecha de entrega la implementación del TFG **no dispone de un plan de pruebas formal ni de pruebas automatizadas**. En su lugar, durante el desarrollo se llevaron a cabo comprobaciones a mano para asegurar la funcionalidad básica del sistema y evitar errores graves.

6.1 Comprobaciones manuales realizadas

Escenario	**Resultado observado**
Alta de nuevo usuario desde la App 2FA	Usuario registrado correctamente en Firestore.
Inicio de sesión con 2FA y aceptación de la solicitud	Cambio de estado a **aceptado** y redirección a pantalla de login exitoso.
Solicitud caducada (timeout 420 s)	Solicitud marcada como **expirada**; app cliente lleva a pantalla de LoginFallido.
Revinculación de dispositivo	deviceId de usuario/{userId} actualizado; el dispositivo anterior deja poder hacer login con esas credenciales.
Concurrencia básica (2 dispositivos simultáneos)	Solo el último dispositivo vinculado puede iniciar sesión para aprobar solicitudes.
Pulsar repetidamente los botones **Login** / **Registro** (> 5 clics en menos de 2 s)	La primera petición se procesa ba correctamente, pero el cambio de pantalla en medio de la respuesta de las otras peticiones hizo que hubiese una desincronización entre la clave CCAB del backend y de las aplicaciones Cliente y 2FA. Se introdujo un cooldown de 3-4 segundos al pulsar estos botones para evitar este error grave.
Intentar acceder a una solicitud cargada en la lista que ha expirado por timeout	Al principio la app 2FA permitía clickar en la solicitud y aceptarla o denegarla sin problema pese a que ya había sido denegada. Se introdujo una verificación de que la solicitud estaba todavía pendiente antes de abrirse al ser clickada

Mantenerse en listado de solicitudes después de revinculación de dispositivo	Originalmente la app 2FA devolvía al usuario a la lista de solicitudes tras aceptar o denegar una solicitud. Esto causaba el serio problema de que, hasta que no cerrase sesión, el dispositivo vinculado previo y el dispositivo vinculado actual podían acceder y gestionar las solicitudes del mismo usuario. Para solucionarlo se cambió la app para que se cerrase sesión automáticamente después de aceptar o denegar una solicitud

Las pruebas se ejecutaron con:

- **Backend** ejecutado en entorno local (Uvicorn+túnel ngrok).
- **App Cliente** y **App 2FA** sobre un emulador de Google Pixel 7 (Android 14) y un Smartphone Samsung Galaxy S25 Ultra (Android 15).

6.2 Limitaciones conocidas de las pruebas

1. **Ausencia de cobertura automatizada:** al no existir suites de *pytest* / JUnit 5, no se puede garantizar que futuras modificaciones no introduzcan regresiones.
2. **Pruebas de carga y seguridad pendientes:** la aplicación no ha sido sometida a escenarios de estrés ni a escaneos de vulnerabilidades.
3. **Entornos limitados:** las pruebas manuales se realizaron sólo en dos dispositivos; no se descartan fallos de compatibilidad.

6.3 Resumen de pruebas

Las comprobaciones manuales proporcionan una confianza razonable para la demostración ante el tribunal, pero **no sustituyen** un marco de validación exhaustivo. La implementación de pruebas automatizadas y mediciones objetivas se identifica como una mejora que desafortunadamente quedará fuera del alcance de la presente entrega.

Estas limitaciones se aceptan dentro del alcance del TFG y se reflejan en las conclusiones como líneas de mejora si el proyecto evolucionara a entorno productivo.

7. Resultados y conclusiones

7.1 Síntesis de los resultados obtenidos

Las comprobaciones manuales descritas en el capítulo 6 confirmaron que **las funciones esenciales del sistema se comportan conforme a lo previsto gracias al protocolo D2OTP** [8][10][11]: alta de nuevos usuarios, inicio de sesión con 2FA, caducidad automática de solicitudes y revinculación de dispositivos funcionaron sin errores críticos.
Estas pruebas también ayudaron a detectar y corregir dos incidencias relevantes:

- una **desincronización de claves** al lanzar múltiples peticiones casi simultáneas, mitigada con un "cool-down" de 3 s en la UI, y
- la posibilidad de operar sobre solicitudes ya expiradas, subsanada añadiendo una validación de estado antes de mostrar el detalle.

Aunque el conjunto fue limitado —dos dispositivos Android y backend local—, el resultado es **un prototipo estable para la defensa**; ningún escenario funcional de los definidos quedó sin cubrir.

7.2 Rendimiento, escalabilidad y disponibilidad

Los requisitos de nivel de servicio fijaban un tiempo p95 inferior a 200 ms en el backend, ≤ 1 s en la app, y un caudal objetivo de 50 peticiones s-1 .
Por falta de un banco de pruebas automatizado, **no se midieron métricas objetivas**; sin embargo, los ensayos manuales muestran que las operaciones completas (M1 → M3) se resuelven en torno a 1–1,5 s (si se tarda más es porque se envían múltiples operaciones al pulsar un botón, como el caso del Login) cuando el backend está en red local y el canal push es inmediato. Con FastAPI asíncrono [19][23] y Firestore gestionado [18], la arquitectura conserva margen para escalar horizontalmente, pero **se requiere un test de carga formal** antes de un despliegue productivo.

7.3 Evaluación de seguridad

El prototipo implementa **cifrado extremo a extremo y autenticación mutua** en cada mensaje gracias a D2OTP [8][10][11], eliminando la exposición de secretos reutilizables y reduciendo el riesgo de phishing o MITM.
No obstante, persisten vectores a reforzar: auditoría externa del algoritmo, dependencia residual del SMS (si se habilita) y la seguridad del dispositivo final. Pese a ello, la combinación de OTP efímeros, canal dual lógico y verificación contextual supone **una mejora ostensible** frente a TOTP [2] o push tradicionales.

7.4 Cumplimiento de objetivos y requisitos

- **O-1 Seguridad extremo a extremo**: cumplido; todas las rutas sensibles viajan cifradas y autenticadas.

- **O-2 Modularidad**: la librería D2OTP se integra en backend y apps sin acoplamientos fuertes.
- **O-3 Escalabilidad horizontal**: diseño sin estado en FastAPI + Firestore, listo para réplicas.
- **O-4 Compatibilidad multiplataforma**: lógica común en KMM; pendiente rama iOS.
- **O-5 Cumplimiento PSD2-SCA**: el usuario ve datos dinámicos antes de aprobar; dos factores diferenciados.
- **O-6 Observabilidad**: cada evento se registra con marcas de tiempo en Firestore.

En términos funcionales, todos los requisitos críticos (R-F1 a R-F3) están presentes; los no funcionales R-NF1 y R-NF2 quedan **condicionalmente cumplidos** a la espera de pruebas de rendimiento.

7.5 Limitaciones identificadas y líneas de mejora

Los capítulos 5 y 6 reflejan las fronteras actuales del prototipo: **ausencia de pruebas automatizadas, versión iOS sin deasarrollar, canal de notificaciones push no implementado, canal SMS no implementado y uso de un túnel ngrok** para exponer el backend.
Las acciones prioritarias son:

1. Desarrollar suites de **pytest/JUnit** y tests de estrés con locust o k6.
2. Migrar el backend a una instancia cloud con **dominio y pinning TLS**.
3. Implementar **push FCM bidireccional** (upstream) e investigar sobre implementación de SMS.
4. Publicar la librería bajo licencia abierta y encargar una **auditoría criptográfica** independiente.
5. Completar la **versión iOS** y habilitar claves de recuperación para pérdidas de dispositivo.

7.6 Conclusión general

El Trabajo de Fin de Grado demuestra la **viabilidad técnica y la pertinencia** de un segundo factor basado en D2OTP para entornos móviles. La solución **cumple los escenarios de negocio previstos**, eleva el nivel de seguridad frente a OTP convencionales sin exigir hardware externo y mantiene una **experiencia de usuario fluida** (aprobación con un toque).

Pese al alcance académico —sin QA industrial ni despliegue global—, el proyecto deja una base sólida y **abierta a evolución**: con pruebas sistemáticas, refuerzo del canal y portabilidad completa, el sistema podría pasar a piloto real y servir como **alternativa ligera a FIDO2**[7] en contextos donde esta aún no es factible.

8. Análisis de Impacto

Durante el desarrollo de este TFG se ha evaluado el **impacto potencial del sistema D2OTP** en diversos ámbitos. A continuación, se analiza cómo esta solución 2FA multiplataforma incide en las dimensiones técnica, de seguridad, social/ética, económica, regulatoria y de sostenibilidad.

8.1 Impacto Técnico

El sistema D2OTP aporta **mejoras técnicas significativas** en autenticación multifactor. En primer lugar, destaca por su naturaleza **multiplataforma e interoperable**. La lógica común de la solución se implementó con Kotlin Multiplatform, permitiendo reutilizar código tanto en Android como (futuramente) en iOS. Esto garantiza una integración homogénea en distintos sistemas operativos y dispositivos, simplificando el mantenimiento y facilitando la portabilidad sin necesidad de reescritura completa. La arquitectura es modular: la librería criptográfica D2OTP se integra en backend y apps cliente **sin acoplamientos fuertes**, actuando como un componente independiente reutilizable. Gracias a esta modularidad, cualquier aplicación puede incorporar D2OTP como segundo factor sin alterar en exceso su propio código, favoreciendo la **interoperabilidad** con diversos entornos.

Otro aspecto técnico clave es la **innovación en el manejo de OTP efímeros y autenticación mutua**. D2OTP se concibe como una solución intermedia que **combina la simplicidad de las OTP simétricas con propiedades avanzadas de autenticación mutua típicas de la criptografía asimétrica**, todo ello **sin requerir infraestructura PKI ni dispositivos de hardware especializados**. En contraste con esquemas tradicionales basados en contraseñas de un solo uso precompartidas, el protocolo genera una **clave OTP efímera por cada mensaje** intercambiado; una vez usado el mensaje, la clave de cifrado derivada se descarta, evitando su reutilización. Esta dinámica efímera proporciona **confidencialidad** a cada intercambio y anula la posibilidad de ataques de repetición o "replay" (un atacante no puede reutilizar un código pasado porque cada OTP es único e irrepetible). Además, D2OTP implementa **autenticación mutua en cada paso**: tanto el cliente como el servidor incluyen en sus mensajes valores criptográficos que el otro debe retornar correctamente, de modo que **ambos extremos se verifican mutuamente sin necesidad de terceros de confianza**. Esta característica, normalmente asociada a protocolos asimétricos avanzados, está lograda aquí con criptografía simétrica, lo cual es un avance técnico importante respaldado por la patente del protocolo. En la práctica, la autenticación mutua garantiza que ni el cliente ni el servidor aceptarán mensajes alterados o provenientes de un impostor.

La **arquitectura del sistema** también demuestra un impacto técnico positivo en términos de **escalabilidad y robustez**. El backend desarrollado con FastAPI es **sin estado** ("stateless") y aprovecha tecnologías cloud como Firebase (Firestore para almacenamiento y FCM para mensajería), lo que permite escalar horizontalmente de forma sencilla añadiendo instancias sin comprometer la consistencia. Este diseño distribuido y escalable soporta

incrementos de usuarios y operaciones, cumpliendo con requisitos de alta concurrencia. De hecho, la solución propuesta fue concebida para ser **horizontamente escalable y asíncrona**, utilizando un servidor web de alto rendimiento y base de datos gestionada, asegurando que pueda atender picos de carga sin degradación significativa.

Por último, la elección de tecnologías multiplataforma y estándares abiertos tiene un impacto técnico positivo en términos de **facilidad de despliegue e integración**. No se necesitan elementos propietarios ni hardware dedicado (como tokens físicos USB o lectores especiales); basta con desplegar la API REST y distribuir la app móvil 2FA a los usuarios. En palabras de la memoria, *"basta integrar la librería D2OTP en el backend (por ejemplo, la API FastAPI) y en la app móvil (Kotlin Multiplatform), apoyándose en Firebase para mensajería; no se precisan llaves U2F ni gestores de certificados"*. Esto supone que una entidad que adopte D2OTP puede hacerlo con un esfuerzo técnico razonable, aprovechando infraestructura existente (internet, notificaciones push) sin montar una PKI propia. En suma, desde la óptica técnica D2OTP es **innovadora, portable y escalable**: introduce un esquema de OTP de un solo uso verdaderamente dinámico y mutual, implementado con una arquitectura moderna y multiplataforma que facilita su adopción en diversos entornos.

8.2 Impacto en la Seguridad

El impacto más notable de D2OTP se observa en la **mejora de la seguridad** de la autenticación. La solución fue diseñada explícitamente para **reducir los riesgos frente a ataques comunes** en esquemas 2FA clásicos, logrando mitigaciones claras contra phishing, ataques de intermediario (MITM), repetición y compromiso de secretos.

En primer lugar, D2OTP ofrece **resistencia robusta al phishing**. En los segundos factores tradicionales (p. ej., códigos OTP SMS o TOTP generados por app), un atacante puede engañar al usuario para que revele o introduzca un código en un sitio falso. Con D2OTP, en cambio, cada petición de autenticación va cifrada y ligada a la identidad de las partes; además, el usuario no tiene que transferir manualmente ningún código. Gracias a la autenticación mutua, el sistema **impide la suplantación de identidad entre las partes comunicantes**, neutralizando intentos de phishing técnico: si un atacante intenta hacerse pasar por el servidor legítimo o interceptar la comunicación, no podrá generar las respuestas criptográficas correctas al no poseer las claves OTP efímeras [10]. Tanto el cliente como el servidor verifican constantemente que el interlocutor es quien dice ser, por lo que se previenen ataques de **Man-in-the-Middle (MITM)** que busquen interponer un intermediario malicioso [10]. De igual modo, **retransmitir o reutilizar códigos resulta inútil**, ya que *"la clave OTP es distinta en cada mensaje y va cifrada"*, haciendo que cada código sea válido sólo en su diálogo específico. Esta propiedad anula los **ataques de replay**, donde un adversario reenvía credenciales capturadas: aunque lograse interceptar un OTP, no podría emplearlo porque ya habría expirado y además estaría cifrado para una sesión concreta. La propia página oficial del protocolo enfatiza que usando exclusivamente **claves de cifrado de un único uso se evitan todos los**

ataques asociados a la reutilización de claves, incluidos **replay, impersonación, descifrado pasivo y análisis criptográfico** [10].

Comparado con esquemas OTP clásicos basados en secretos estáticos (HOTP/TOTP), D2OTP elimina varios vectores de ataque conocidos. Por ejemplo, en TOTP la semilla secreta residen tanto en el servidor como en el dispositivo; si dicha semilla se filtra, un atacante podría generar OTP válidos ilimitadamente. En D2OTP **nunca se expone la clave base compartida** ni viaja ningún secreto reutilizable, dificultando enormemente que un compromiso de un dispositivo derive en la clonación del segundo factor. Adicionalmente, los OTP de D2OTP están **cifrados en tránsito** y ligados a datos de la operación, a diferencia de un código SMS o de una aplicación genérica que podría ser interceptado en texto plano. Esto significa que interceptar la comunicación (ej. capturando un mensaje push) no es suficiente para el atacante, pues los valores de autenticación viajan cifrados y sólo el destinatario legítimo puede descifrarlos. En resumen, **romper la confidencialidad de un canal de transporte no compromete la autenticación**, ya que la seguridad no depende únicamente del canal (se opera un **doble canal lógico** cifrado más allá de la red móvil).

Cabe mencionar que D2OTP también aborda ataques más sutiles como la **aprobación involuntaria (approval fraud)**. Dado que la aplicación autenticadora muestra al usuario información contextual de la operación (monto, destino, etc.) antes de permitir aprobar, se dificulta que un usuario autorice por error una transacción fraudulenta. Este principio de **"firma dinámica"** cumple con las recomendaciones regulatorias para evitar engaños por ingeniería social, haciendo que el usuario pueda detectar solicitudes anómalas antes de confirmarlas. La **verificación contextual** (el usuario ve exactamente qué está autorizando) aumenta la eficacia del segundo factor frente a ataques de tipo consentimiento engañado, que han obligado a muchas instituciones a mejorar la contextualización de sus notificaciones 2FA.

El impacto en seguridad queda reflejado en la evaluación realizada: el prototipo implementado logró **cifrado de extremo a extremo y autenticación mutua en cada mensaje**, *"eliminando la exposición de secretos reutilizables y reduciendo el riesgo de phishing o MITM"* según las conclusiones obtenidas. Esto supone una **mejora sustancial frente a enfoques 2FA convencionales** como TOTP o los códigos push tradicionales. En pruebas manuales, ningún escenario de ataque común logró comprometer la comunicación gracias a las protecciones de D2OTP. En definitiva, la solución eleva notablemente el nivel de seguridad de la autenticación de doble factor en comparación con esquemas tradicionales, cerrando vectores críticos de phishing, MITM y replay.

Por supuesto, es importante señalar que **ningún sistema es invulnerable** y D2OTP no es excepción. Persisten ciertas consideraciones: por ejemplo, la seguridad depende en última instancia de mantener seguro el dispositivo autenticador. Si un atacante tuviera acceso físico o malware en el teléfono donde reside la app 2FA (especialmente si estuviera desbloqueado), podría potencialmente aprobar operaciones fraudulentas. Estas situaciones extremas (compromiso del dispositivo de usuario) **no son completamente solucionables sólo con el protocolo** y requieren medidas adicionales (como

PIN o biometría en la app 2FA y educación al usuario para no aprobar notificaciones inesperadas). De hecho, en la memoria se identifica como vector a reforzar la *"seguridad del dispositivo final"* y la necesidad de una auditoría criptográfica externa del algoritmo para garantizar que no existan debilidades no descubiertas. Con todo, las mejoras introducidas (OTP efímeros, doble canal lógico, autenticación mutua, confirmación contextual) **suponen un salto cualitativo en seguridad** respecto a 2FA tradicionales. D2OTP logra un equilibrio óptimo: aproxima el nivel de protección de soluciones como FIDO2/WebAuthn (resistentes al phishing y con no repudio) pero usando solo criptografía simétrica, lo que permite que **funcione en dispositivos que no soportan hardware especial U2F/passkeys**. Esto amplía el alcance de una **autenticación robusta a entornos donde las soluciones más complejas (p. ej. llaves físicas o certificados digitales) no son factibles**, llenando un vacío importante en la seguridad práctica.

8.3 Impacto Social y Ético

En el plano social y ético, D2OTP contribuye positivamente a la **protección de los usuarios** y a la accesibilidad de mecanismos de alta seguridad sin dificultar la experiencia de uso. Uno de los principales impactos es la **mejora de la privacidad del usuario**. El sistema fue concebido bajo principios de "privacy by design": **minimiza la recopilación y almacenamiento de datos personales**. Por ejemplo, no almacena secretos sensibles a largo plazo en servidores públicos; únicamente se guardan hashes de contraseñas y metadatos imprescindibles para la operación, cumpliendo con los principios de minimización y limitación de finalidad del GDPR. Esto significa que el usuario no entrega información privada adicional más allá de la necesaria para autenticarse, reduciendo su exposición. Al evitar el uso de canales inseguros como SMS (que podrían filtrar datos personales o números de teléfono a atacantes) y no requerir documentos de identidad digitales ni certificados vinculados a su identidad, D2OTP **respeta la privacidad** intrínsecamente. En esencia, el usuario mantiene el control de sus credenciales (que nunca abandonan su dispositivo en forma aprovechable por terceros) y la plataforma sólo maneja datos cifrados y referencias internas, reforzando la confianza de que su información personal no será indebidamente explotada.

Otro impacto social relevante es la **facilidad de uso y accesibilidad** de la solución, lo cual tiene implicaciones éticas al posibilitar seguridad para un público amplio sin discriminación por recursos. A diferencia de algunos segundos factores que exigen dispositivos físicos dedicados (tarjetas de coordenadas, *tokens* RSA, llaves USB, etc.) o conocimientos técnicos elevados, D2OTP se basa en la **utilización de dispositivos que los usuarios ya poseen** (su smartphone) y en interacciones muy sencillas. La aprobación de una solicitud 2FA se realiza con **un solo toque en la app móvil**, sin tener que transcribir códigos manualmente. Esta simplicidad disminuye la fricción para el usuario final, facilitando la adopción del segundo factor en lugar de que sea visto como un obstáculo. Una experiencia de usuario fluida es crucial: si las medidas de seguridad son demasiado engorrosas, muchos usuarios tienden a evitarlas. D2OTP, al ofrecer una confirmación inmediata vía notificación

segura, **aumenta la probabilidad de que los usuarios acepten y utilicen el 2FA**, mejorando la seguridad colectiva del sistema. De hecho, el proyecto logró mantener *"una experiencia de usuario fluida (aprobación con un toque)"* incluso elevando el nivel de seguridad respecto a OTP convencionales, lo que indica un **equilibrio adecuado entre seguridad y usabilidad**.

La **accesibilidad económica y tecnológica** de D2OTP redunda también en beneficios sociales. Al no requerir hardware adicional, se elimina la barrera de entrada que supondría que cada usuario tuviera que adquirir o portar un dispositivo de autenticación extra. Esto es importante desde un punto de vista ético: la seguridad no debe ser un lujo disponible solo para quienes pueden costear dispositivos especiales, sino un derecho al alcance de todos los usuarios. Con D2OTP, cualquier persona con un teléfono inteligente básico puede tener segundo factor de alta seguridad, sin tener que depender de tokens físicos costosos o infraestructura propietaria. Además, la ausencia de requisitos especiales facilita su despliegue masivo en contextos como servicios gubernamentales o aplicaciones de gran base de usuarios, promoviendo una **inclusión digital segura**. En escenarios corporativos, los empleados no necesitan gestionar múltiples credenciales o aparatos, sino simplemente autorizar en su móvil, lo que mejora la aceptación y reduce errores (como perder un token físico o transcribir mal un código).

Desde la perspectiva de la **confianza del usuario**, el sistema también aporta beneficios. Saber que la autenticación de dos pasos implementada incluye mecanismos avanzados (cifrado integral, validación mutua) puede generar en el usuario una **mayor confianza en la plataforma que utiliza** (banca en línea, comercio electrónico, etc.). Los usuarios tienden a sentirse más seguros realizando operaciones sensibles si perciben que existe una protección robusta contra fraudes. En D2OTP, al mostrarse los detalles de la operación en la app autenticadora antes de aprobar, se le da transparencia al usuario sobre qué está autorizando, empoderándolo para detectar intentos maliciosos. Esta transparencia y control refuerzan la **conciencia y educación en seguridad**: el usuario aprende a verificar los datos de cada petición y a no aprobar solicitudes que no reconozca, convirtiéndose en parte activa de la defensa. Éticamente, este enfoque es preferible a sistemas opacos donde el usuario aprueba sin entender, ya que aquí se fomenta una relación más informada y de colaboración en la seguridad entre el usuario y el sistema.

En suma, el impacto social de D2OTP es proporcionar **seguridad elevada con mínima carga para el usuario**, protegiendo su privacidad y mejorando la confianza en servicios digitales. Esto contribuye a un entorno en que más usuarios adopten MFA (reduciendo fraudes a gran escala) y donde incluso personas sin conocimientos avanzados puedan beneficiarse de autenticación fuerte. Reducir los incidentes de suplantación y robo de cuentas no solo protege a individuos, sino que también fortalece la confianza colectiva en las transacciones electrónicas, un bien social en la era digital.

8.4 Impacto Económico

La adopción de D2OTP como solución de autenticación de segundo factor conlleva **impactos económicos favorables**, tanto en costos de implementación y operación como en ahorros por reducción de fraude.

En términos de **coste de adopción e infraestructura**, D2OTP representa una alternativa **más económica** frente a esquemas tradicionales basados en hardware o certificados. Al ser una solución puramente de *software* que aprovecha dispositivos existentes (los smartphones de los usuarios) y canales de comunicación ya desplegados (internet, servicios push), **evita inversiones en dispositivos físicos dedicados y en su distribución**. Por ejemplo, muchas organizaciones han provisto en el pasado *tokens* físicos o tarjetas a sus usuarios para MFA, lo cual implicaba comprar, gestionar inventarios, reemplazar unidades perdidas/dañadas y soportar su logística. Con D2OTP, todo ese costo desaparece: el "token" es una aplicación móvil. De hecho, las **credenciales software (soft tokens)** como la que propone D2OTP *"son generalmente mucho menos costosas de desplegar y gestionar en comparación con tokens hardware"*, resultando ideales cuando se necesita 2FA sin incurrir en gastos de equipos de seguridad [31]. Un token físico puede costar entre decenas de euros por usuario (sumando fabricación y soporte) frente a un costo prácticamente nulo por usuario de una app móvil, aparte del desarrollo inicial de la plataforma. Asimismo, **añadir nuevos usuarios** es trivial y no conlleva gastos materiales – basta con registrar el dispositivo en el sistema – a diferencia de los hard tokens donde escalar a miles de usuarios supone compras masivas y complejas distribuciones [31].

D2OTP también **ahorra costes en infraestructura de seguridad** al prescindir de elementos costosos como una PKI interna o servidores de autenticación especializados. No se necesitan certificados digitales emitidos para cada usuario ni módulos hardware de seguridad (HSM) para custodia de claves privadas, ya que el esquema opera con claves simétricas efímeras gestionadas entre las partes. Esto simplifica la arquitectura y **reduce gastos de mantenimiento** asociados a renovar certificados, auditorías de PKI, etc. El protocolo evita incluso la necesidad de complejos *handshakes* de establecimiento de sesión (tipo TLS) en cada operación, ya que desde el primer mensaje se transmite información cifrada segura sin terceros [10]. Al usar algoritmos simétricos eficientes (AES, HMAC) en lugar de operaciones de criptografía asimétrica pesada, el sistema **minimiza el consumo de CPU y memoria por autenticación** [10]. Esto implica que, para un mismo servidor, se pueden manejar más autenticaciones por segundo con D2OTP que con un esquema basado en RSA/ECC o firmas digitales, antes de necesitar escalamiento vertical. En las pruebas, el prototipo mostró que con un backend sencillo era posible resolver una solicitud completa en ~1 segundo, quedando la mayor parte del tiempo en espera de la aprobación del usuario. Con un diseño asíncrono y servicios gestionados, **la arquitectura tiene margen para escalar horizontalmente sin grandes inversiones**, pudiendo atender decenas de peticiones por segundo en entornos cloud estándar. Por tanto, desde la perspectiva de la operación continua, D2OTP **requiere menos recursos de proceso y comunicación** para lograr comunicaciones seguras, lo cual *"disminuye la latencia operativa e incrementa el número de operaciones*

resueltas usando los mismos recursos" disponibles [10]. En resumen, su eficiencia técnica se traduce en **eficiencia de costos**: más seguridad con menos infraestructura.

El otro gran componente económico es el **ahorro por prevención de fraude**. Los ataques de phishing y suplantación con credenciales robadas generan cada año pérdidas multimillonarias a empresas e individuos (transferencias fraudulentas, acceso a sistemas críticos, robo de datos, etc.). Al mejorar drásticamente la resistencia a estos ataques, D2OTP puede **reducir significativamente los costes asociados al cibercrimen**. Estudios citados en la documentación de D2OTP estiman que en 2025 el costo global del cibercrimen superará los 10 billones de dólares, y que hasta un 80% de esos delitos podrían originarse en ataques de phishing [10]. Si se logra atajar el phishing mediante autenticación robusta, *"la prevención efectiva del phishing podría reducir los costos globales del cibercrimen en varios billones de euros anuales"* [10]. En este sentido, D2OTP tendría un **impacto económico macro** ayudando a evitar una porción de esas pérdidas colosales, gracias a su mecanismo de defensa que impide la suplantación incluso cuando los usuarios caen en engaños. A nivel micro, para una entidad financiera por ejemplo, implementar D2OTP puede reducir drásticamente el fraude en cuentas de clientes (p. ej. transferencias no autorizadas) y, con ello, **evitar pérdidas directas de dinero** que la entidad a menudo termina asumiendo para resarcir al cliente. También se evitan costes indirectos: investigaciones forenses, sanciones regulatorias por brechas, incrementos en primas de ciberseguros, y daño reputacional que impacta la confianza de los clientes (difícil de cuantificar, pero real). Así, el **retorno de inversión (ROI)** de una solución como D2OTP podría medirse en el dinero que **no** se pierde por ataques frustrados. La propia patente afirma que D2OTP *"proporciona una rentabilidad económica inmediata tras su despliegue"*, atribuyendo ahorros equivalentes a las pérdidas económicas y reputacionales evitadas por impedir phishing/MITM [10].

En conclusión, la adopción de D2OTP conlleva **ahorros y beneficios económicos** en múltiples frentes: minimiza gastos en hardware y mantenimientos complejos, aprovecha de forma óptima los recursos existentes (infraestructura y dispositivos), y reduce la exposición a fraudes costosos. Para organizaciones grandes, esto puede traducirse en millones de euros ahorrados tanto en CAPEX/OPEX de seguridad como en mitigación de fraudes. Incluso para usuarios finales o pequeñas empresas, evita costes asociados (por ejemplo, ya no es necesario pagar por mensajes SMS de verificación, que a gran escala suponen sumas importantes, ya que D2OTP puede usar notificaciones push prácticamente gratuitas). En suma, D2OTP **democratiza el acceso a una autenticación fuerte de bajo coste**, a la vez que actúa como una herramienta de **reducción de pérdidas por ciberataques**, contribuyendo positivamente a la economía digital segura.

8.5 Impacto Regulatorio

El ámbito regulatorio de la seguridad informática y, en particular, de la autenticación, es cada vez más exigente. En este contexto, D2OTP tiene un

impacto muy favorable, ya que **ayuda al cumplimiento de estándares y normativas vigentes en materia de autenticación fuerte**.

Uno de los referentes normativos clave es la Directiva PSD2 de la Unión Europea y sus estándares técnicos asociados de **Autenticación Reforzada de Clientes (Strong Customer Authentication, SCA)** para pagos electrónicos. PSD2 exige a los proveedores de pagos implementar una autenticación de dos factores sólida y, especialmente, un mecanismo de **vinculación dinámica** de la operación (dynamic linking) que asocie el código de autenticación con el importe y el beneficiario de la transacción, mostrando dichos datos al usuario [32]. D2OTP fue diseñado teniendo estos requisitos en mente. El protocolo cumple con la definición de **autenticación robusta de dos factores** (combina algo que el usuario tiene – su dispositivo con las claves OTP – y algo que sabe/es – su PIN o biometría si se habilita, o su contraseña base para iniciar la petición). Más aún, **implementa la vinculación dinámica sin necesidad de certificados digitales**: cada código OTP generado es único por transacción y va ligado criptográficamente a los datos específicos de esa operación, los cuales son presentados al usuario antes de aprobar. Como indica la memoria, *"la confirmación mutua y el vínculo entre mensajes pueden transportar datos dinámicos de la operación, satisfaciendo la exigencia de firma dinámica sin certificados"*, cumpliendo así con lo exigido por PSD2 SCA. En la validación del prototipo se comprobó que el usuario ve en la app 2FA la cantidad y destino (u otros detalles) de la operación real antes de aprobar, y que cualquier alteración de esos datos haría que el código no validase. De este modo, una institución financiera podría utilizar D2OTP para **dar conformidad a PSD2** de forma sencilla, sin adquirir soluciones propietarias: el sistema ya *"implementa una autenticación reforzada y hace uso de un código de enlace para cada transacción"*, facilitando la adaptación de sus sistemas a los requisitos regulatorios de pagos en línea [10]. Esto no solo evita posibles sanciones por incumplimiento, sino que también brinda seguridad jurídica al adoptar una tecnología alineada con las directrices europeas.

Otra referencia importante son los estándares de seguridad en el desarrollo de aplicaciones móviles, como la guía **OWASP Mobile Application Security Verification Standard (MASVS)**. Durante este proyecto se han seguido buenas prácticas coherentes con estos estándares, lo que imprime un impacto regulatorio positivo en términos de conformidad con frameworks de la industria. Por ejemplo, se incorporaron controles como **almacenamiento seguro de claves en el Keystore del dispositivo** y cifrado de datos en reposo en Firestore, minimizando la exposición de datos sensibles (de acuerdo con requerimientos de OWASP MASVS para protección de datos locales). También se implementó un completo **registro de eventos de seguridad** (altas de dispositivo, mensajes M1/M2/M3, etc.) con sellos de tiempo para auditoría y no repudio, cumpliendo con recomendaciones de trazabilidad y monitoreo de OWASP MASVS v2. De hecho, uno de los objetivos planteados (O-6) fue dotar de observabilidad al sistema siguiendo pautas de OWASP, lo que se logró satisfactoriamente. Adicionalmente, el modelo de amenazas desarrollado cubrió las top 10 amenazas móviles de OWASP, asegurando que se consideraran los vectores más relevantes de acuerdo con la comunidad de seguridad. Esto ubica a D2OTP en un **contexto de cumplimiento de**

estándares de facto, algo valioso de cara a auditorías o certificaciones futuras de la aplicación móvil.

En entornos fuera de Europa, D2OTP igualmente aporta cumplimiento a lineamientos de seguridad. Por ejemplo, el NIST en EE.UU. en su publicación SP 800-63B recomienda evitar segundos factores basados en SMS debido a vulnerabilidades en la red telefónica; D2OTP se alinea con esta recomendación al privilegiar canales cifrados de internet (push) y claves dinámicas en vez de SMS. Del mismo modo, D2OTP ofrece un nivel de seguridad que cumple o excede lo requerido por normativas sectoriales que exigen MFA (por ejemplo, normas bancarias locales, PCI-DSS en el caso de tarjetas, o lineamientos de autenticación gubernamental). Su flexibilidad también podría permitir adaptarse a criterios de **eIDAS** (identidad digital europea) si se integrase como mecanismo de firma/autenticación de nivel sustancial, aunque esto no fue objeto directo del TFG.

En resumen, **el impacto regulatorio de D2OTP es facilitar el cumplimiento normativo en materia de autenticación fuerte**, proporcionando de serie las características necesarias para adherirse a leyes como PSD2 y alineándose con estándares reconocidos (OWASP MASVS, OWASP Top 10, NIST). Las organizaciones que implementen D2OTP estarán mejor preparadas para pasar auditorías de seguridad y demostrar a reguladores y clientes que cuentan con medidas de protección de última generación. Esto les evita incurrir en soluciones costosas para cumplir mandatos legales, ya que con una única implementación cumplen múltiples requisitos (doble factor, datos dinámicos, registros auditables, etc.). En un panorama donde las exigencias legales sobre seguridad tienden solo a incrementarse, D2OTP supone una **herramienta proactiva de cumplimiento** que aporta tranquilidad regulatoria.

8.6 Sostenibilidad

Finalmente, es importante considerar el impacto de D2OTP en la **sostenibilidad**, tanto desde la perspectiva ambiental como de la sostenibilidad técnica a largo plazo. En este apartado, D2OTP muestra ventajas destacables al ser una solución ligera en recursos y con mínima dependencia de elementos físicos.

Desde el punto de vista **medioambiental**, D2OTP contribuye a **reducir la huella de hardware y residuos electrónicos** asociados a la seguridad. Al prescindir de *tokens* físicos, tarjetas o dispositivos dedicados, evita la fabricación de miles (o potencialmente millones) de aparatos electrónicos adicionales si se desplegara masivamente. Esto tiene un efecto no menor: la producción y descarte de dispositivos contribuye significativamente a la contaminación; por ejemplo, en 2019 el mundo generó **53 millones de toneladas métricas de residuos electrónicos** (e-waste) de dispositivos diversos [33]. Cada *token* de autenticación física eventualmente se convierte en desecho (ya sea por fin de vida útil de la batería o por obsolescencia), sumándose a ese flujo de e-waste. D2OTP, al utilizar smartphones ya existentes como segundo factor, **prolonga la utilidad de dispositivos en posesión del usuario** en lugar de introducir nuevos gadgets de un solo propósito. Esto se traduce en menos consumo de materiales (plásticos,

metales) y energía en fabricación, y menos residuos a gestionar al final del ciclo. Incluso considerando la energía consumida por la aplicación móvil, ésta es marginal en comparación con la energía y emisiones implicadas en producir hardware dedicado para MFA y distribuirlo globalmente. Por tanto, implementar D2OTP en una organización en lugar de repartir llaves físicas U2F o tarjetas OTP contribuye a una **política más ecológica**, reduciendo la necesidad de transporte y logística de dispositivos, así como la huella de carbono asociada.

En cuanto a **eficiencia energética y de recursos informáticos**, D2OTP está diseñado para ser **ligero**. Las operaciones criptográficas principales son simétricas (AES-256, HMAC SHA-256) que son computacionalmente eficientes incluso en dispositivos móviles modestos. No requiere prolongados cálculos asimétricos ni grandes intercambios de datos para establecer confianza, dado que evita *handshakes* con certificados [10]. En concreto, *"hace uso de claves de cifrado simétricas, menores consumidoras de recursos que las asimétricas…; no necesita de la ejecución previa de un proceso de enlace (handshake) con certificado digital; desde el primer mensaje enviado se puede enviar información cifrada"* [10]. Esto significa que cada autenticación con D2OTP usa **pocos ciclos de CPU** y muy poco tráfico de red (unos cuantos kilobytes a lo sumo, al ser mensajes breves). En servidores, esta eficiencia permite atender más solicitudes por unidad de energía consumida, mejorando la **sostenibilidad energética** del servicio. En los clientes, el impacto en la batería del teléfono es prácticamente despreciable, ya que la app 2FA sólo trabaja unos instantes al llegar una notificación (el cifrado de los mensajes M1-M3 es muy rápido). Así, D2OTP **minimiza el consumo de recursos en cada operación de comunicación** al requerir menos mensajes intercambiados y menos cómputo que otras alternativas [10]. Esto no solo es bueno para la experiencia del usuario (menos espera, menos gasto de datos móviles), sino que a gran escala reduce la carga en infraestructuras de comunicaciones y data centers. En suma, la solución es **eficiente** en el uso de hardware y energía, cualidad valiosa en un mundo donde las tecnologías deben escalar de forma sostenible.

La **sostenibilidad técnica** y en mantenimiento de D2OTP también merece mención. Al estar construido con componentes modernos y mantenibles (Kotlin multiplataforma, Python, APIs REST estándar), el sistema es más fácil de actualizar y adaptar con el tiempo. La base de código compartida entre plataformas reduce la duplicidad y el esfuerzo futuro de introducir mejoras o parches de seguridad, ya que una corrección en la librería D2OTP se propaga a todas las plataformas soportadas. Esto implica un **mantenimiento más sencillo y menos propenso a errores**, contribuyendo a la longevidad del proyecto. La arquitectura modular significa que partes del sistema pueden reemplazarse o escalarse sin necesidad de rehacer todo: por ejemplo, si en un futuro se quisiera cambiar Firebase por otro backend, o agregar otro método de notificación, es factible hacerlo manteniendo el núcleo D2OTP intacto. Esta flexibilidad se traduce en **sostenibilidad evolutiva**: el sistema puede crecer y adaptarse a nuevas necesidades con menor costo de desarrollo. Un ejemplo de esto es la capacidad del protocolo para incorporar nuevos algoritmos criptográficos según se requiera, incluso pensando en la **era post-cuántica**; el diseño de D2OTP es adaptable para integrar cifrados resistentes a

computación cuántica cuando sea necesario [10]. Esto garantiza que la solución no quede obsoleta con la evolución de las amenazas, sino que puede actualizarse para seguir siendo segura en el largo plazo.

Finalmente, la ausencia de dependencias en terceros de confianza o licenciamiento costoso hace a D2OTP más sostenible desde el punto de vista operativo. Al no basarse en un servicio externo de autenticación (más allá de usar un canal de mensajería genérico), las organizaciones pueden autogestionar la solución sin riesgo de verse afectadas por la discontinuación de un proveedor. Si el proyecto se libera como código abierto (algo propuesto en la memoria), la comunidad podría contribuir a su mejora continua, aumentando su sostenibilidad comunitaria y transparencia, lo cual también enlaza con consideraciones éticas de confianza colectiva en la herramienta.

En conclusión, D2OTP demuestra un impacto positivo en sostenibilidad: es **ambientalmente responsable** al eliminar la necesidad de nuevo hardware (reduciendo residuos y emisiones), es **energéticamente eficiente** al requerir mínimos recursos computacionales por autenticación, y es **técnicamente durable** gracias a su diseño flexible y mantenible. En un momento donde se busca que las soluciones tecnológicas no solo sean seguras sino también sostenibles, D2OTP ofrece una aproximación que equilibra seguridad de vanguardia con responsabilidad en el uso de los recursos.

9. Bibliografía

[1] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache y O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm," RFC 4226, IETF, dic. 2005. Disponible: https://datatracker.ietf.org/doc/html/rfc4226

[2] D. M'Raihi, S. Machani, M. Pei y J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, IETF, may 2011. Disponible: https://datatracker.ietf.org/doc/html/rfc6238

[3] Frontegg, "*What Is a Time-Based One-Time Password (TOTP)?*," Frontegg Blog, Apr. 2025. Disponible: https://frontegg.com/blog/what-is-a-time-based-one-time-password-totp

[4] M. Borrelli *et al.*, "*Mitigating real-time relay phishing attacks against mobile push and OTP-based MFA*," M.S. thesis, James Madison University, 2024. .Disponible : https://commons.lib.jmu.edu/masters202029/4/

[5] P. Grassi, M. Garcia y J. Fenton, *Digital Identity Guidelines* (SP 800-63-3), NIST, jun. 2017. Disponible: https://doi.org/10.6028/NIST.SP.800-63-3

[6] World Wide Web Consortium, "Web Authentication: An API for accessing Public Key Credentials Level 1," Recomendación W3C, 4-mar-2019. Disponible: https://www.w3.org/TR/webauthn-1/

[7] FIDO Alliance, *Client to Authenticator Protocol (CTAP) v2.1*, feb. 2020. Disponible: https://fidoalliance.org/specs/fido-v2.1-rd-20200902/ctap2.1-rd-20200902.html

[8] J. A. de la Vega Crespo y J. Carrillo Verdún, "Sistema para el cifrado y autenticación de comunicaciones con autenticación mutua de los comunicantes," Patente WO 2022/018310 A1, 27-ene-2022. Disponible: https://patentscope.wipo.int/search/es/detail.jsf?docId=WO2022018310

[9] European Banking Authority, "Regulatory Technical Standards on Strong Customer Authentication and Secure Communication under PSD2," Jun. 2017. Disponible: https://www.eba.europa.eu/regulation-and-policy/payment-services-and-electronic-money/regulatory-technical-standards-on-strong-customer-authentication-and-secure-communication-under-psd2

[10] D2OTP, "Innovación en la Ciberseguridad de las Comunicaciones," d2otp.com, 2025. Disponible: https://www.d2otp.com/

[11] FEINDEF Brokerage Event, "Protocol for secure communications D2OTP," b2match.com, May 2025. Disponible: https://www. b2match.com/e/feindef-2025/opportunities/UGFydGljaXBhdGlvbk9wcG9ydHVuaXR5OjE1OTQyNA%3D%3D

[12] Google Cloud, "Server-side encryption | Firestore in Native mode," Cloud Documentation, 2025. Disponible: https://cloud.google.com/firestore/native/docs/server-side-encryption

[13] OWASP Foundation, "OWASP Mobile Top 10 – Final List 2016," 2016. Disponible: https://owasp.org/www-project-mobile-top-10/

[14] Fortinet, “What is Threat Modeling? How does it Work?” Fortinet Cyber-Glossary, 2025. Disponible: https://www.fortinet.com/resources/cyberglossary/threat-modeling

[15] National Institute of Standards and Technology (NIST), “APP-10: Poorly Implemented Cryptography,” Mobile Threat Catalogue, Disponible: https://pages.nist.gov/mobile-threat-catalogue/application-threats/APP-10.html

[16] Agilicus, “NIST SP 800-63B: How Well Do I Know You?” Blog post, Dec. 29, 2021. Disponible: https://www.agilicus.com/nist-sp-800-63b-how-well-do-i-know-you/

[17] P. A. Grassi, J. L. Fenton, N. A. Lefkovitz, K. M. Greene, and W. E. Danker, “*Digital Identity Guidelines: Authentication and Lifecycle Management*,” NIST Special Publication 800-63B, Jun. 2017. Disponible: https://doi.org/10.6028/NIST.SP.800-63b

[18] Google Cloud, “Firestore in Native mode documentation,” Google Cloud, 2025. Disponible: https://cloud.google.com/firestore/native/docs

[19] S. Ramírez, “FastAPI documentation,” FastAPI, 2025. Disponible: https://fastapi.tiangolo.com/

[20] JetBrains, “*Kotlin Multiplatform Mobile Developer Guide*,” JetBrains Documentation, 2025. Disponible: https://kotlinlang.org/docs/multiplatform-mobile-getting-started.html

[21] OWASP Foundation, “*Mobile Application Security Verification Standard (MASVS) v2.0*,” Apr. 2023. Disponible: https://owasp.org/www-project-mobile-security-testing-guide/mstg/mstg-introduction/0x04-masvs.

[22] ngrok, “*Agent ingress and TLS termination*,” *ngrok Documentation*, 2025. Disponible: https://ngrok.com/docs/agent/ingress/

[23] S. Ramírez, “*Run a Server Manually*,” *FastAPI Documentation*, 2025. Disponible: https://fastapi.tiangolo.com/deployment/server-manually/

[24] Google Cloud, “*Customer-managed encryption keys (CMEK) | Firestore*,” *Firebase Documentation*, 2025. Disponible: https://firebase.google.com/docs/firestore/security/cmek

[25] Firebase, “*Get started with Cloud Firestore Security Rules*,” *Firebase Documentation*, 2025. Disponible: https://firebase.google.com/docs/firestore/security/get-started

[26] Firebase, “*FCM ports and your firewall*,” *Firebase Documentation*, 2025. Disponible: https://firebase.google.com/docs/cloud-messaging/concept-options#ports_and_your_firewall

[27] Google, “Use the REST API | Cloud Firestore,” Firebase Documentation, 2025. Disponible: https://firebase.google.com/docs/firestore/use-rest-api

[28] Firebase, “Concepts & Options | Cloud Messaging,” Firebase Documentation, 2025. Disponible: https://firebase.google.com/docs/cloud-messaging/concept-options

[29] JetBrains, “Kotlin Multiplatform Mobile (KMM),” Kotlin Documentation, 2025. Disponible: https://kotlinlang.org/lp/mobile/

[30] Firebase, “Firebase Documentation,” Google Developers, 2025. Disponible: https://firebase.google.com/docs

[31] Telnyx, *“Hard token vs. soft token: What’s the difference?,”* Telnyx Blog, 2023. Disponible: https://telnyx.com/resources/hard-token-vs-soft-token

[32] Authy (Twilio), *“Understanding Dynamic Linking in PSD2,”* Medium, Jun. 2019. Disponible: https://medium.com/@Authy/understanding-dynamic-linking-in-psd2-352d75bce8b3

[33] L. Kugler, *“The Battle to Mitigate E-Waste,” Communications of the ACM*, vol. 68, no. 6, pp. 16–18, Jun. 2025. Disponible: https://cacm.acm.org/news/271073

10.Anexos

Anexo 1 - Plan de Trabajo de este TFG:
https://drive.google.com/file/d/1OQCzG-tqv8EDEQ1c5wjoXW0xYhfl3Xff/view?usp=sharing

Anexo 2 - Implementación de la App 2FA:
https://github.com/LPerezL/AppAuthD2OTP

Anexo 3 - Implementación de la App Cliente:
https://github.com/LPerezL/AppBancaFalsa

Anexo 4 - Implementación del Backend API:
https://github.com/LPerezL/API_REST

Anexo 5 – Librería D2OTP Kotlin: Contenida como module dentro tanto de la implementación de la App Cliente como la App 2FA, se llama libreriaD2OTPKotlin

Anexo 6 – Librería Python: Contenida como archivo .py dentro tanto de la implementación de Backend API, se llama libreriaD2OTPPython

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 21:07:01 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)