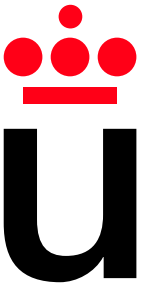

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería del Software

Curso 2023-2024

Trabajo Fin de Grado

ESTUDIO E IMPLEMENTACIÓN DE UN ESQUEMA DE PSI (PRIVATE SET INTERSECTION) DESCENTRALIZADO

Autor: Santiago Arias Paniagua
Cotutora: Ana Isabel Gómez Pérez
Cotutor: Cristian González García

Agradecimientos

Gracias a mi familia por su apoyo durante el desarrollo del proyecto.

Gracias también a mis tutores Ana Isabel Gómez Pérez y Cristián González García por su orientación, su conocimiento y su ayuda durante el desarrollo y redacción del proyecto.

Resumen

La privacidad de los datos es una preocupación fundamental para todas las personas. Uno de los objetivos de la seguridad en los sistemas actuales es proteger la confidencialidad mientras se procesan datos de especial protección, bien sea en nuestros dispositivos, o en servidores externos. La intersección de conjuntos privados, o *Private Set Intersection (PSI)*, es una técnica que permite que las partes involucradas en una comunicación, puedan comprobar si comparten elementos en sus conjuntos de datos sin revelar información extra. El objetivo del proyecto es desarrollar un sistema multiplataforma que permita evaluar protocolos destinados al cálculo de la intersección de conjuntos privados. En esta versión del sistema se han implementado criptosistemas seleccionados de la literatura en base a sus capacidades homomórficas, Esto significa que permitirán realizar operaciones matemáticas sobre datos cifrados.

El sistema desarrollado se basa en una red descentralizada de servicios web, que exponen una API REST y dispositivos Android, que cuentan con su propia aplicación. Los servicios web están implementados en el lenguaje Python, y la aplicación Android emplea Kotlin en lo referente a actividades, adaptadores y servicios, y Java para las interfaces, objetos e implementaciones.

Mediante esta plataforma hemos obtenido resultados iniciales sobre el rendimiento de los criptosistemas propuestos en dispositivos de bajos recursos.

Palabras clave:

- Android
- Ciberseguridad
- Esquemas criptográficos de cifrado homomórfico
- Criptosistemas Paillier y Damgård–Jurik
- Intersección de conjuntos privados
- Mensajería descentralizada
- Privacidad

Índice de contenidos

Índice de tablas

Índice de figuras

Índice de códigos

1 Introducción

Las organizaciones e individuos intercambian información a un ritmo sin precedentes, por lo que cobra más importancia que nunca que no se revele más información de la necesaria, para respetar nuestro derecho a la privacidad, en un mundo en el que cada vez parece más complicado. Desde finales del siglo XIX se identifica este derecho, con estas palabras "every such case the individual is entitled to decide whether that which is his shall be given to the public" [1]. En la era digital actual, la seguridad y la privacidad no deben interferir en la experiencia del usuario, siendo transparentes y a la vez proporcionando los mecanismos necesarios para preservarlas. Ejemplos son las recientes claves de acceso en los gestores de contraseñas [2], que validan que somos quienes decimos ser; o los certificados de seguridad, que validan la autenticidad de la información a la que estamos accediendo (como una página web [3]). Estos mecanismos nos permiten transmitir información sensible de forma segura a través de internet. Esto representa un reto importante para quienes conocemos cómo funciona el software.

Las técnicas criptográficas son la forma en la que protegemos la información y aseguramos su integridad y confidencialidad. Su uso requiere protocolos de cifrado que conviertan la información en archivos ininteligibles para cualquier persona que no tenga la clave correspondiente para descifrarla. Estos protocolos se basan en problemas matemáticos complejos, como puede ser el problema de calcular las clases del enésimo residuo como Paillier, o la factorización de números grandes como en RSA. Estos "problemas"permiten realizar fácilmente las operaciones de cifrado, es decir, las operaciones en un sentido; pero muy difícil en el sentido contrario, es decir, intentar descifrar sin la clave.

La Computación Segura Multiparte (SMPC) es la denominación de una fa-

milia de protocolos que ofrecen el cálculo de una función matemática sobre una serie de entradas de datos que tienen que permanecer privadas y ofreciendo solo el resultado al final. En este trabajo, nos centramos en un problema particular que es el cálculo de intersecciones privadas, *PSI* por sus siglas en inglés - *Private Set Intersection* [4]. PSI es una técnica criptográfica que permite aprender los elementos comunes que hay en dos conjuntos cifrados, de manera que no se revele información adicional que no se pueda deducir a partir solo de la intersección [5]. Este problema ha atraído la atención de muchos investigadores debido a su amplia variedad de aplicaciones, contribuyendo a la proliferación de muchos enfoques diferentes.

Airdrop, que es el servicio de intercambio de archivos sin conexión de Apple, integrado en todos sus dispositivos, tiene errores de diseño en su protocolo que permite a posibles atacantes aprender los números de teléfono y direcciones de correo de los dispositivos [6]. El problema de Airdrop recae en que mientras el dispositivo escanea, envía el hash SHA-256 en claro de su teléfono y dirección de correo, en una forma de peticiones de descubrimiento. En un mundo ideal, un dispositivo que tiene dicho número o dirección entre sus contactos, comprueba esa petición y le manda un mensaje confirmando que se conocen. En el mundo real, un atacante puede monitorizar todas las peticiones y, dada la poca entropía de un número de teléfono, deshacer el hash del mismo. El problema va más allá, por el hecho de tener una dirección de correo guardada en la agenda, se podría estar enviando el hash del número de teléfono a una persona con la que se ha estado en contacto, pero que por motivos de privacidad no se les ha dado el teléfono, de esta forma, se podría obtener igualmente. En países con niveles altos de censura como China, el gobierno ha aprovechado esta vulnerabilidad para interceptar a personas que intercambian archivos que se consideran censurados [7]. La comunidad científica critica a Apple por no haber incluido la mejora que se propuso en su momento, pero esto, en mi opinión, es un indicativo de la importancia de la investigación en este ámbito, puesto que implica cambiar el protocolo que millones de personas utilizan de forma cotidiana.
Este problema de privacidad ejemplifica a la perfección la necesidad de la Computación Segura Multiparte, y en particular, la necesidad de PSI, así como la necesidad de continuar la investigación en el ámbito.

El desarrollo del internet de las cosas (IoT) requiere que estos esquemas sean seguros pero también eficientes en dispositivos heterogéneos que cuentan con recursos limitados. Es necesaria una arquitectura que permita la implementación para la intersección de conjuntos privados basada en la necesidad de interoperabilidad, escalabilidad, seguridad, rendimiento y relevancia en aplicaciones del mundo real. Una arquitectura similar, y gran inspiración para el proyecto es la investigación del uso de la plataforma Midgar [8], con diferentes combinaciones algorítmicas (RSA, AES y SHA3) y una propuesta de cómo la criptografía puede ser utilizada en los mensajes intercambiados entre nodos para crear redes seguras IoT. En ella, se abordan aspectos como confidencialidad, integridad, autentica-

ción, no repudio y privacidad, y se evalúan combinaciones con el objetivo de encontrar el mejor equilibrio entre consumo de recursos y seguridad.

El desafío de este tipo de arquitecturas es la complejidad de la concurrencia, la red descentralizada y el registro de datos. Estos retos permiten complementar las demostraciones matemáticas de seguridad tradicionales con un uso más realista de la técnica en dispositivos con recursos limitados. Mientras que este proyecto no se enfoca directamente en dispositivos IoT, sí que lo hace en las capacidades de comunicación seguras entre dispositivos heterogéneos en una red descentralizada, y asegura que los datos privados permanezcan protegidos en cualquier sistema.

Para limitar el alcance del proyecto y que sea posible abarcarlo en un tiempo razonable, se ha decidido implementar dos esquemas criptográficos para la intersección de conjuntos privados: Paillier [9] y Damgård-Jurik [10]. Se ha optado por realizar dos implementaciones para comparar su rendimiento y seguridad. La arquitectura del sistema permite la extensión a otros esquemas en el futuro, con cambios mínimos requeridos en el código fuente.

El proyecto presenta varias ramas que lo hacen interesante. Por un lado, la aplicación de Android, y por otro, la API REST e interfaz de los servicios web, que proporcionan al usuario una interfaz sencilla para probar esquemas criptográficos. La propuesta de implementación contribuye al campo de la criptografía, permitiendo una futura extensibilidad por otros desarrolladores, quienes podrán probar sus implementaciones criptográficas en una red robusta entre dispositivos heterogéneos. Este sistema ofrece una forma de demostrar vulnerabilidades y medir tiempos, lo cual puede beneficiar a la comunidad criptográfica y a posibles implementaciones productivas de software basadas en estos principios. Además, se propone un sistema de registros robusto que reporta cada evento relevante a una base de datos, permitiendo la extracción y explotación de la información.

La justificación de este proyecto se basa en que hay poco contenido similar al propuesto, hay investigaciones similares[11][12] e incluso propuestas de arquitecturas IoT seguras[13], pero es un campo que no se suele abordar frecuentemente, dejando de lado los dispositivos cada vez más presentes en nuestras vidas, como teléfonos móviles o dispositivos IoT[8]. Algunos trabajos han explorado el uso de dispositivos de bajas prestaciones, como Arduino, para probar conceptos de IoT, aunque sin centrarse en la seguridad, de hecho, muchos de ellos suelen carecer de ella[14].

2

Contexto y alcance del proyecto

En este capítulo se presentan los objetivos del proyecto, el estudio del estado del arte y la planificación y metodología llevada a cabo para su realización.

El objetivo es implementar un sistema multiplataforma en el que poder calcular conjuntos de intersecciones privados entre dispositivos heterogéneos. En este caso se ha utilizando una arquitectura basada en servicios web creados con Python y una aplicación de Android. La inclusión de implementaciones en Android proporciona mediciones más representativas de dispositivos cotidianos.

Un objetivo secundario es el estudio de técnicas como la Evaluación Polinomial Oculta, por sus siglas en inglés *OPE - Oblivious Polynomical Evaluation* [5, 15], una aproximación que puede ser útil para dispositivos IoT [16], y una variante de cardinalidad basada en la evaluación polinómica. Es otro objetivo comparar sus rendimientos, y poder determinar con exactitud si utilizar esta aproximación para la privacidad es una buena opción.

A parte de la implementación de las técnicas y los esquemas criptográficos, el trabajo también se centra en crear una red descentralizada, fiable y potente para dispositivos de bajos recursos, en la que potencialmente se puedan probar más implementaciones, con interfaces atractivas, usables y accesibles, y casos de uso extensibles.

2.1. Objetivos del proyecto

Los objetivos de este proyecto se resumen en los siguientes puntos:

- Revisar el estado del arte de PSI y de la criptografía homomórfica.
- Desarrollar un sistema multiplataforma extensible que permita la evaluación de diferentes esquemas criptográficos.
- Evaluar el rendimiento de intersecciones de conjuntos privados utilizando esquemas que permitan calcular intersecciones con distintas técnicas de forma descentralizada.
- Implementar de forma completa los criptosistemas de Paillier y Damgård–Jurik capaces de realizar las operaciones homomórficas propias de estos sistemas.
- Recopilar y analizar los resultados del sistema desarrollado.
- Documentar todo el proyecto para facilitar la comprensión, el mantenimiento y la posible ampliación de éste.

2.2. Descripción del problema

El problema abordado se centra en la necesidad de calcular intersecciones de conjuntos privados en un entorno digital, donde la seguridad y privacidad de los datos son aspectos críticos. Para entender mejor la situación, pongamos un ejemplo: cuando instalas una aplicación de mensajería o redes sociales, éstas habitualmente piden acceso a tus contactos, es importante que la información total de tus contactos no sea desvelada. En este caso podríamos aplicar la intersección privada de conjuntos para compartir únicamente los contactos que estén registrados en la aplicación que se va a utilizar.

Este ejemplo, que explica en la práctica lo que es el PSI, sirve de base para establecer cuál es el problema a tratar. Es un protocolo que involucra dos partes, y que cada parte tiene información que ni necesita, ni quiere, compartir con la otra parte o con una autoridad central. La definición es la siguiente:

Alice y Bob tienen su conjunto de datos o conjuntos "secretos"

$$A = \{a_1, \cdots, a_n\},$$

$$B = \{b_1, \cdots, b_m\}.$$

Hay que tener en cuenta que los elementos son cadenas de «bits» con poca entropía porque suelen tener una estructura muy marcada, como por ejemplo

números de teléfono, apellidos o palabras de un determinado idioma. Alice quiere enterarse de los elementos que tiene en común con Bob, $A \cap B$, sin revelarle ninguna información adicional en el proceso. Dependiendo de las variantes seleccionadas para implementar, se puede hacer que Bob tampoco pueda conocer detalles de la intersección, habiendo varios escenarios como se puede ver en la Tabla 2.1.

	$A \cap B$	$\lvert A \cap B \rvert$	$\lvert A \rvert$	$\lvert B \rvert$
Alice	a	a	✓	d
Bob	b	b	c	✓

Tabla 2.1: Modelo de compartición de la información entre Alice (Cliente) y Bob (Servidor) donde a, b, c y d son bits cualesquiera limitados por la restricción $a \vee b =$ `True` (es decir, al menos uno conoce la intersección)

Alice y Bob necesitan de una técnica que les permita comprobar la intersección de sus elementos comunes, este es el problema que se va a abordar.

Hay que mencionar que teóricamente se ha demostrado que no existen protocolos seguros para PSI en cualquier variante de información desvelada (cualquiera de las dos partes o ambas descubren la intersección; se revela o no para cada parte el tamaño del conjunto ajeno) en un escenario en el que un atacante tenga acceso a recursos computacionales infinitos [17]. En el escenario más sencillo tanto Alice como Bob conocen el tamaño de su entrada, pero en algunos escenarios esta información debe ser mantenida en secreto.

2.3. Estudio de la situación actual

Grandes empresas tecnológicas usan PSI en sus proyectos y actividades: Apple ha desarrollado un sistema de detección de Material de Abuso Sexual Infantil *(Child Sexual Abuse Material, CSAM*, que utiliza PSI para comparar imágenes cargadas por los usuarios con una base de datos de imágenes CSAM conocidas [18]. Aunque no dan detalles de implementaciones, su sistema PSI está bien documentado [19] y, aunque Apple decidió dar marcha atrás con el plan [20], era una idea muy innovadora. No es la primera vez que Apple utiliza PSI en sus dispositivos, y es que su gestor de contraseñas también hace uso de PSI para alertarte si alguna de tus contraseñas se han visto comprometidas. En este caso, se calcula la intersección contra una base de datos de $1{,}5 \cdot 10^9$ expuestas [21]. Como se comentó anteriormente, Airdrop también fue un intento de PSI, aunque se descubriera la inseguridad del protocolo por el intercambio de hashes subyacente.

Otras empresas, como Meta, han realizado sus propios desarrollos. Por ejemplo *Private Data Lookup (PDL)* permite a los usuarios consultar de forma privada

un conjunto de datos en el servidor a la hora de crear contraseñas o de descubrir contactos de manera privada [22].

2.3.1. Métodos, técnicas y variantes de PSI

En esta subsección se investigarán distintos métodos, técnicas, esquemas, protocolos y algoritmos utilizados para realizar PSI, cada uno con su funcionamiento y debilidades. Aunque este trabajo se centra en los métodos basados en criptografía homomórfica, la elección de un método depende del nivel de privacidad requerido y el consumo computacional que se pueda permitir.

Antes de comenzar me gustaría recalcar que nos basamos en modelos de adversarios "semi-honestos", aunque los protocolos sean correctos, no quita que sin medidas adicionales, controles, terceras partes de confianza, o pruebas que comprueben que alguna parte no está siendo engañosa, las partes pudieran desviarse del protocolo de manera no maliciosa, o pueden darle ventajas no deseadas, sin poder subvertir el protocolo en sí. Están diseñados para funcionar bien incluso si las partes no son completamente confiables, pero no es un modelo de seguridad perfecto [23].

Variante Private Matching (PM)

Este esquema fue presentado en el artículo *Efficient Private Matching and Set Intersection* [5]. Requiere criptografía homomórfica y se basa en la evaluación polinómica oculta o Oblivious Polynomial Evaluation (OPE). El artículo se basa en el esquema criptográfico de Paillier, explicaremos la implementación como PSI.

Existen dos conjuntos secretos definidos asociados a dos partes A y B, como en la descripción del problema: $A = \{a_1, \cdots, a_n\}$ y $B = \{b_1, \cdots, b_m\}$. Nos propone el siguiente proceso:

1. A calcula los coeficientes de un polinomio utilizando como raíces los elementos de su conjunto y cifra los resultados:

$$p(A) = \prod_{i=1}^{n} (X - a_i) = \sum_{i=0}^{n} p_i X^i, \ \forall i = 0, \cdots, n.$$

   Calcula $E(p_i)$ y se lo envía a B.

2. B recibe el conjunto cifrado y la clave pública de A. Evalúa el polinomio anterior con los elementos de su conjunto aprovechando el cifrado homomórfico, calcula $E(p(b_j))$, $\forall j \in 0, \cdots, m$.

3. Tomando una r aleatoria, B calcula utilizando la clave pública de A:

$$E\left(r \cdot p\left(b_j\right) + b_j\right)$$

4. A recibe el resultado de las evaluaciones E_j y descifra la intersección $D_j = D\left(E_j\right), \forall j \in 0, \cdots, m$. Entonces puede comprobar si $D_j \in A$.

Pero, ¿cómo funciona realmente? Cuando evaluamos un polinomio con una raíz, el resultado es cero, y como es una técnica que se apoya en el homomorfismo del criptosistema, ocurrirá lo siguiente:

$$D\left(E\left(r \cdot p\left(b_j\right) + b_j\right)\right) = b_j \Leftrightarrow p\left(b_j\right) = 0 \Leftrightarrow b_j \in A$$

B no sabe si lo que ha evaluado es un cero o no, por lo que no se le ha revelado ninguna información. Es importante que las operaciones homomórficas que haga B utilicen la clave pública de A, puesto que la que debe poder descifrarlo es la clave privada de A. Por otro lado, este algoritmo requiere que la criptografía sea probabilística mediante el valor r, de lo contrario, todos los cifrados de "cero"tendrían la misma representación y se habría revelado la intersección a B y a un posible atacante. Una posible debilidad en el protocolo es que existe probabilidad no nula de que haya una colisión si $b \notin X$ tal que $E\left(r \cdot p\left(b\right) + b\right)$ coincide con el cifrado de un elemento de X.

La seguridad y vulnerabilidad de la técnica en este modelo funciona siempre que las partes cumplan correctamente la especificación, y se tenga un cifrado semánticamente seguro. En un escenario semi honesto, ¿que pasaría si la parte A ejecutase el algoritmo, siendo su parte todos los valores posibles? Si $A = \mathcal{U}$, A obtendría todos los datos del servidor. Una solución es limitar el grado del polinomio, denegando las propuestas que no cumplan con el cardinal del conjunto, o utilizar terceras partes de confianza.

Variante de cardinalidad (PM_c) Si solo queremos saber el tamaño de la intersección, sin querer calcularla como tal, podemos hacer lo siguiente:

$$E\left(r \cdot p\left(b_j\right) + 0^+\right)$$

Cada vez que la parte A reciba una cadena de ceros, significa que un elemento está en la intersección, por ello lo puede contar. Es importante cambiar el orden bien de la evaluación o del resultado, para que no se revelen datos posicionales, y que el solicitante no pueda hacer inferencias en caso de que los valores estén bajo el mismo dominio.

Intersecciones calculadas según el dominio

Una alternativa cuando se trabaja con dispositivos IoT o bajo un dominio de datos concretos es calcular las intersecciones según el dominio. No se basa en un

algoritmo concreto, pero aprovecha que el criptosistema sea homomórfico, que debe ser probabilístico como en el caso de *OPE*. La idea de este protocolo ha sido modificada para que terceras partes no puedan inferir valores y está basada en un artículo que propone una multiplicación por cero o por uno [24].

El proceso es el siguiente:

1. A y B comparten un mismo dominio de esos datos, esto quiere decir que:

$$A = \{a_1, \cdots a_n\} \text{ donde } A \subset D$$

$$B = \{b_1, \cdots b_m\} \text{ donde } B \subset D$$

2. A cifra un 0 o un 1 los elementos del dominio, dependiendo de si están presentes en su conjunto de datos, podemos considerar:

$$f_A(x) = \begin{cases} E(0) & \text{si } x \notin A \\ E(1) & \text{si } x \in A \end{cases} \quad \forall x \in D$$

   Después le envía el resultado y su clave pública a B.

3. B recibe los datos y para cada elemento del conjunto ordenado recibido, cifrará un 0 con la clave pública de A o multiplicará el valor del elemento correspondiente homomórficamente por el escalar dos:

$$f_B(x) = \begin{cases} E(0) & \text{si } x \notin B \\ E(a_n \cdot 2) & \text{si } x \in B \end{cases} \quad \forall x \in |A|$$

   Al terminar, le envía de vuelta los resultados a A.

4. A descifra los datos, y donde encuentre un 2, significa que dicha posición está presente en ambos conjuntos de datos:

$$Dec_A(x) = 2 \text{ si } x \in A \cap B$$

$$Dec_A(x) = 0 \text{ si } x \in A \setminus B \text{ o } x \notin A \cup B$$
$$\forall x \in |A'|$$

Puede convertirse en algo ineficiente si el conjunto de datos sobre el que se opera es grande, el esquema criptográfico es lento por naturaleza, o se utilizan claves de más longitud.

Esta técnica revela más información (A sabrá de antemano que todos los elementos que no intersecten, no están en B) al trabajar sobre un dominio, A puede saber que si B tiene más elementos, estarán acotados al resto de los que no estén en A. Como en el resto de técnicas, si $A = D$, A conocería todo el conjunto

de datos de B al recibir el cálculo. Si se utiliza un esquema determinista, el cifrado de cero siempre será el mismo, lo mismo para 1 y 2, si un atacante se hace con el mensaje enviado, sabría automáticamente qué elementos del dominio tienen A y parte de B, ya que el mensaje que se envía representa el orden en el dominio. En uno probabilístico no pasa porque no coinciden los valores cifrados de un mismo número, garantizando que una tercera parte no sabe qué elementos son ceros, unos o doses en cualquier conjunto cifrado.

Es una técnica poco eficiente si se trabaja con dominios grandes, si se tiene un conjunto de 50 datos, pero un dominio de 500, se estarán cifrando 450 valores innecesariamente. Un caso de uso IoT es un termostato inteligente que regula la temperatura de una casa con sensores distribuidos por las habitaciones. Si no quiero que estos datos sean visibles en mi tráfico de red, podría aplicar esta técnica usando un dominio de temperaturas, entre los sensores y el termostato.

Otras variantes

Otras variantes de PSI que se han revisado pero no se van a abordar en este documento son APSI ó SI Autenticado[25], que garantiza que las partes no pueden mentir con sus conjuntos de entrada y MP-PSI [26] ó Multi parte en el que más de 2 partes pueden computar la intersección

2.3.2. Criptografía homomórfica

Los esquemas criptográficos homomórficos son aquellos que permiten realizar operaciones aritméticas sin tener que descifrar los datos previamente. Permiten realizar algunas operaciones con valores cifrados, y otras con tanto valores cifrados como escalares. Para aclarar este concepto, supongamos que tenemos $A = 10$ y $B = 20$, siendo $E_h(A) = c_A$ y $E_h(B) = c_B$. Con un esquema de cifrado homomórfico, por ejemplo, se cumple lo siguiente:

$$Dec(c_A + 20) = 30,\ Dec(c_A + c_B) = 30,\quad Dec(c_A * 2) = 20$$

Destacar que esto *es solo una visualización*, los criptosistemas no suman y multiplican así, sino que usan técnicas aritméticas y consiguen el resultado representado anteriormente, como veremos.

Este proyecto se centra en la criptografía homomórfica, e implementa el esquema Private Matching - PM, por lo que estos criptosistemas son necesarios para demostrar su funcionamiento.

Hay que destacar que los criptosistemas homomórficos tienen diferentes niveles de capacidad de procesamiento de operaciones sobre datos cifrados, pueden ser:

1. **Cifrado parcialmente homomórfico (PHE).** Soporta la suma o multiplicación de datos cifrados, pero no ambas, y de forma ilimitada. Ejemplos son el criptosistema de Paillier [27] y el Damgård–Jurik [10], son aditivamente homomórficos, pero no permiten multiplicar textos cifrados. También los hay multiplicativos como ElGamal[28]. Se pueden utilizar para crear sistemas de votación [29].

2. **Cifrado algo homomórfico (SHE).** Permite realizar un número limitado de operaciones de suma y multiplicación, siendo la limitación la profundidad de las operaciones o la cantidad de operaciones que se pueden realizar hasta que la seguridad del cifrado quede comprometida o el sistema inservible. Introducen ruido en las operaciones, distorsión que se añade a los datos, pudiendo hacer que los datos descifrados no sean precisos. Un ejemplo es el esquema *Yet Another Somewhat Homomorphic Encryption* (*YASHE*)[30].

3. **Cifrado completamente homomórfico (FHE).** Soporta un número ilimitado de sumas y multiplicaciones sobre los datos cifrados, tiene el mayor nivel de funcionalidad de los tres esquemas, pero son exigentes a nivel computacional. Aquí se encuentran los esquemas Brakerski-Gentry-Vaikuntanathan (BGV) [31], Cheon, Kim, Kim and Song (CKKS) [32] o Brakerski-Fan-Vercauteren (BFV) [33]. Se pueden considerar una extensión de los algo homomórficos en los que se utilizan técnicas para controlar el crecimiento del ruido[34].

A continuación se muestra cómo funcionan los criptosistemas de Paillier, de Damgård–Jurik y de Brakerski-Fan-Vercauteren.

Paillier

Paillier es un criptosistema de clave pública que tiene las propiedades de ser homomórfico, aditivo, y probabilístico. Su seguridad se basa en la dificultad de calcular residuos para números grandes, de forma similar al problema RSA pero en este caso basado en el denominado problema ≪Decisional composite residuosity assumption≫ (DCRA). En concreto, la dificultad computacional del problema se establece en que dado un número entero z y un parámetro no primo n, hay que decidir si existe un y tal $z = y^n \mod n^2$ [27].

Las siguientes operaciones se detallan tras consultar trabajos [9, 27, 35, 36] y la librería de Python *phe* [37]. A continuación resumimos los principales pasos:

Generación de claves: Para esto, se eligen dos primos "grandes" (p y q) a partir de los cuales se calcula $\lambda = \text{mcm}(p-1, q-1)$, donde la función mcm representa el mínimo común múltiplo. Para la clave pública se toma $n = pq$, y $g \in \mathbb{Z}^*_{n^2}$, esto es en el conjunto de enteros que son relativamente primos con

n^2, haciendo módulo con n^2. Por lo general y por simplicidad se suele utilizar $g = n + 1$.

La clave pública es (n, g) y la privada es (λ, μ). Se define este último parámetro como el inverso multiplicativo:

$$\mu = \left(L\left(g^{\lambda} \mod n^2\right)\right)^{-1} \mod n \text{ donde } L(u) = \frac{u-1}{n}$$

Este último parámetro se utilizará en el descifrado.

Cifrado: Para poder cifrar un mensaje $m \in \mathbb{Z}_n$ se siguen estos pasos:

1. Se selecciona un número aleatorio $r \in \mathbb{Z}_n^*$.
2. El cifrado de m será $c = g^m \cdot r^n \mod n^2$

La selección de un número cifrado es lo que hace que el sistema sea probabilístico, ya que dos textos no tienen por qué tomar el mismo valor al cifrarse.

Descifrado: Para descifrar c entonces se realiza la operación $m = L(c^{\lambda}) \cdot \mu$, con los módulos adecuados.

Suma homomórfica de números cifrados Partimos de dos números cifrados con la misma clave pública c_1 y c_2. La suma homomórfica se consigue multiplicando los textos cifrados:

$$c_{suma} = c_1 \cdot c_2 \mod n^2$$

Suma homomórfica de un número cifrado y un escalar Se quiere sumar un número cifrado c y un escalar k, para ello es necesario conocer la clave pública, de forma que luego se pueda descifrar. La fórmula es:

$$c_{sumak} = c \cdot (g^k \mod n^2) \mod n^2$$

Lo que se hace esencialmente es "cifrar" el número pero sin el factor aleatorio, simplificando la operación cuando solo queremos añadir un escalar, sin perder seguridad.

Multiplicación de un número cifrado y un escalar Matemáticamente, para un escalar k seguimos una fórmula simple, similar a las anteriores:

$$c_{producto} = c^k \mod n^2$$

Todas estas fórmulas anteriores no vienen de la nada, sino que descomponiendo los mensajes cifrados c se puede comprobar fácilmente, en el caso de la

multiplicación, que si sustituimos el valor del cifrado, tendríamos:

$$c_{producto} = (g^m \cdot r^n)^k \mod n^2$$

Distribuyendo la potencia, podemos obtener:

$$c_{producto} = g^{m \cdot k} \cdot r^{n \cdot k} \mod n^2$$

Esta operación tiene inconvenientes: para el caso con $k = 0$, todo número elevado a 0 es 1, si fuera necesario realizar esta operación, sería conveniente cifrar un 0 directamente. Otro problema sería multiplicar con $k = 1$, el resultado sería el mismo, por lo que lo recomendable sería cifrar un 0 y sumarle ese valor homomórficamente.

Estas propiedades pueden ser extendidas para soportar operaciones más complejas sobre valores cifrados, siempre que las operaciones fundamentales de suma y multiplicación sean suficientes para expresar esas nuevas operaciones.

Damgård–Jurik

Damgård–Jurik es una generalización del criptosistema de Paillier. A diferencia de Paillier, que tiene un espacio de cifrado $\mathbb{Z}_{n^2}$, Damgård–Jurik amplía el espacio de cifrado a $\mathbb{Z}_{n^{s+1}}$, donde s es un entero positivo. Esto quiere decir que Damgård–Jurik proporciona una mayor flexibilidad pudiendo parametrizar s, y una mayor seguridad al trabajar en espacios de cifrado mayores. Esto hace que abarque un mayor abanico de aplicaciones en criptografía que Paillier. Las siguientes operaciones se basan en el papel oficial [10], en la librería de Python *damgard-jurik* [38] y en las similitudes que comparte con Paillier.

Generación de claves: El proceso es similar al de Paillier, incluyendo ahora el parámetro del factor de expansión s:

De nuevo se eligen dos primos (p y q) a partir de los cuales se calcula $\lambda = \text{mcm}(p-1, q-1)$, donde la función mcm representa el mínimo común múltiplo. Para la clave pública se toma $n = pq$, y $g \in \mathbb{Z}^*_{n^{s+1}}$, que cumpla con una condición adicional de $g = (1+n)^j \cdot x \mod n^{s+1}$ para dos parámetros j y x, que pueden ser seleccionados adecuadamente de forma aleatoria.

La clave pública es (n, g, s) y la privada (λ, μ). Se define este último parámetro como el inverso multiplicativo:

$$\mu = \left(L\left(g^{\lambda} \mod n^{s+1}\right)\right)^{-1} \mod n \text{ donde } L(u) = \frac{u-1}{n}$$

Se puede ver que si $s = 1$, entonces es el criptosistema de Paillier

Cifrado: En este caso se quiere cifrar $m \in \mathbb{Z}_{n^s}$, de forma similar a como se hace con Paillier:

1. Se selecciona un número aleatorio $r \in \mathbb{Z}_n^*$
2. Se calcula el cifrado de m como $c = g^m \cdot r^{n^s} \mod n^{s+1}$

Descifrado: Para descifrar c, seguimos los siguientes pasos:

1. Se calcula $L(u) = L(c^\lambda \mod n^{s+1})$
2. Se recupera m, aplicando el algoritmo de desencriptación de Paillier de forma recursiva.

Podemos ver que se mantienen las propiedades del criptosistema anterior, como ser probabilístico, y es fácil usar las mismas operaciones (suma homomórfica de números cifrados y suma y multiplicación con escalar), trabajando en este caso en el nuevo espacio de cifrado con módulos n^{s+1}.

Brakerski-Fan-Vercauteren (BFV)

Este esquema ofrece un cifrado homomórfico completo [33], que permite sumar y multiplicar en el dominio cifrado. Su problema se basa en el aprendizaje con errores [39] y tiene variantes que se basan en el anillo de aprendizaje con errores [40]. Las operaciones sobre datos cifrados introducen mucho ruido en el resultado, y son necesarias técnicas como el *Bootstrapping* para intentar mantener la coherencia de los textos planos en presencia de ruido. También en operaciones homomórficas como la multiplicación, requiere funciones referidas de relinearización para mantener el texto cifrado en el dominio adecuado.

Aunque la complejidad de este esquema es superior a los esquemas anteriores, se ha incluido a modo de prueba de concepto para realizar comparaciones con un esquema totalmente homomórfico. Hasta ahora es un esquema que no se ha roto por la computación cuántica, pudiéndose considerar muy seguro. Las variantes del diseño sobre anillos, así como las sumas y productos hace que se puedan realizar cálculos complejos sobre los datos cifrados. Aunque es un sistema complejo, se dará una idea de cuál es el comportamiento subyacente. Las siguientes operaciones se basan en la interpretación del papel oficial [33] y en la librería py-fhe [41] que implementa el esquema.

Generación de claves: El criptosistema depende de la creación tres polinomios, $s(x), a(x), b(x)$. La clave pública es la pareja $(a(x), b(x))$ y la privada es $s(x)$. Sin entrar en detalles técnicos, los parámetros t, q, n nos darán los posibles valores de los coeficientes y grados de esos polinomios.

1. Se generan los parámetros (t, q, n), en función de un parámetro de seguridad que representa el número de operaciones necesario para romperlo.

2. Se generan los polinomio secreto $s(x) \in \mathbb{Z}_q[x] / (x^n + 1)$, un polinomio de error $e(x)$ y un polinomio aleatorio $a(x) \in \mathbb{Z}_q[x] / (x^n + 1)$.

3. Se calcula el polinomio público (forma parte de la clave pública) a partir de los polinomios anteriores $b(x) = -a(x) \cdot s(x) + e(x) \mod q$.

Cifrado: Es importante desatacar que dada la naturaleza del sistema, los mensajes se tienen que convertir en un polinomio, este paso se refiere como «encoding». Una vez realizado este paso el mensaje $m(x) \in \mathbb{Z}_t[x] / (x^n + 1)$. Se seleccionan polinomios aleatorios $r_1(x), r_2(x), r_3(x)$ con coeficientes relativamente bajos. Se calcula el cifrado como:

$$c_0(x) = b(x) \cdot r_1(x) + r_1(x) + m(x) \mod q$$

$$c_1(x) = a(x) \cdot r_1(x) + r_2(x) \mod q$$

El cifrado es la pareja de polinomios $(c_0(x), c_1(x))$. Es necesario que la t sea menor que la q seleccionada anteriormente, ya que sino el grado del polinomio a generar sería superior, y las limitaciones del sistema impiden que se cifre un mensaje con un módulo menor que el mensaje a cifrar, se saldría del espacio de cifrado sobre el que se está trabajando.

Descifrado: Partiendo de la pareja $(c_0(x), c_1(x))$ se puede realizar calculando

$$c(x) = c_0(x) + c_1(x) \cdot s(x)$$

El mensaje de ese resultado sería el texto plano, podría considerarse $m(x) = f^t_{redondeo}(c(x))$, donde $f^t_{redondeo}$ representa una función que redondea los coeficientes entero más cercano, necesaria por la introducción de ruido en el cifrado y operaciones homomórficas.

Implementarlo requiere un alto conocimiento de criptografía avanzada y la teoría de números que escapa el alcance de este proyecto.

2.4. Planificación del proyecto

Aquí se presenta cómo se ha planificado y gestionado el proyecto, así como la metodología que se ha seguido para abordarlo. El seguimiento del proyecto se ha realizado con reuniones mediante Teams y con dos repositorios de GitHub, uno

para el desarrollo de Android[1], y otro para el desarrollo de Python[2].

2.4.1. Metodología

La gestión del proyecto con dos desarrollos en paralelo, se ha utilizado una aproximación de Modelo Incremental [42]. Se ha planificado de forma que las dependencias de los dos sistemas estuvieran alineadas durante el desarrollo.

Para un mejor seguimiento del proyecto se ha utilizado un tablero "Kanban", para poder tener de un vistazo las tareas pendientes, realizadas o bloqueadas. La idea principal era que una vez se validasen los desarrollos realizados, estos quedasen integrados en la línea base, y es así como ha ido evolucionando el proyecto.

Se han planteado ocho iteraciones con desarrollo involucrado, sumando una primera dedicada al estudio y a los requisitos de más alto nivel del sistema, y una última dedicada a la experimentación y pruebas. Como se establecieron las reuniones de Teams cada 14 días, para planificar este proyecto se previeron diez iteraciones de 14 días. Esto se puede ver en la tabla 2.2.

Esta planificación estima el tiempo disponible para dedicar al proyecto intercalando una jornada laboral de 6 horas y dos clases por cuatrimestre, con lo que sus trabajos conllevan, esto dependerá también de la fecha estimada para las iteraciones y la carga de trabajo esperada. Estas iteraciones han variado en el tiempo por motivos ajenos al desarrollo, como Navidad o Semana Santa, siendo algunas más cortas y otras más largas.

Iteraciones	Trabajo estimado (horas)
Iteración 1: 14 días - 17/11/2023	40
Iteración 2: 14 días	22
Iteración 3: 14 días	19
Iteración 4: 14 días	21
Iteración 5: 14 días	9
Iteración 6: 14 días	27
Iteración 7: 14 días	23
Iteración 8: 14 días	15
Iteración 9: 14 días	33
Iteración 10: 14 días	19
Total estimado	228

Tabla 2.2: Planificación de las iteraciones - Modelo incremental

Los requisitos principales de alto nivel se definieron en la primera reunión y lo primero que se ha hecho para realizar el proyecto ha sido tener una estima-

[1]Repositorio Android: https://github.com/4rius/APP_PSI
[2]Repositorio Python: https://github.com/4rius/WS_PSI

ción de tiempo para realizar los estudios pertinentes con respecto a los conceptos que se van a abordar. Esto incluye la investigación de redes descentralizadas, de esquemas criptográficos y su factibilidad para incorporarlos en el proyecto, así como investigaciones similares. De aquí sale la hoja de ruta que se iría refinando a medida que se avanza con las iteraciones. Es una estimación conservadora, teniendo en cuenta que unas iteraciones duren más que otras o que surjan problemas inesperados.

Usando la filosofía del modelo incremental, se han hecho aportaciones periódicamente a la documentación, a medida que avanzaba el desarrollo. Estas aportaciones incluyen nuevos análisis, que van pasando de requisitos de más alto nivel en las primeras iteraciones a requisitos refinados. Con cada iteración, el análisis se revisa y actualiza para reflejar los nuevos requisitos identificados. De igual forma, se parte de un diseño de alto nivel, o incluso una idea, sobre el que se van identificando nuevas necesidades. Más detalles de estas previsiones se pueden encontrar en el Apéndice A.1.

2.4.2. Desarrollo o implementación del software y pruebas

Los incrementos estables se han realizado periódicamente, y en base a lo que se descubre, y la retroalimentación de los tutores, se van haciendo ajustes. Como usando el modelo incremental, pueden surgir cambios de alcance en el sistema, esta tabla de tiempos puede no cumplirse como se espera. Mi conocimiento de desarrollo Android era mayor y conocía la realización de algunos requisitos, luego la Tabla 2.3 muestra que este desarrollo se prevé menor que el de Python, en el que nunca había desarrollado una aplicación del estilo.

Tarea	Iteración	Trabajo (horas)
Elección y pruebas de librerías de criptografía	2	2
Elección y pruebas de tecnología para la red descentralizada	1	2
Configuración proyecto y base de datos en Firebase	6	5
Desarrollo Android	-	47
Desarrollo Python	-	60
Total desarrollo		**116**

Tabla 2.3: Planificación del desarrollo general

La planificación detallada de Android y Python se encuentra en las Tablas A.3 y A.4 de los anexos, respectivamente.

Mientras se está desarrollando, se realizan pruebas periódicas de carácter manual, el objetivo es probar casos límites, estresar al sistema, o encontrar formas de romperlo. Cuando se validan los incrementos, se realizan pruebas de regresión, para comprobar que el resto del sistema sigue funcionando. La estimación para estas pruebas se puede encontrar en la Tabla A.5 de los anexos.

Una vez realizada la implementación de ambos sistemas, se utilizan como base para los experimentos. Se planificaron unas 29 horas en los experimentos, como se puede ver en la Tabla A.6 de los anexos.

2.4.3. Cierre de proyecto

Como muestra la Tabla 2.4, antes de entregar del proyecto, se revisa la implementación del sistema, el experimento realizado y la documentación redactada. No lleva iteración porque no se realiza ningún incremento, está estimado para realizarse después de la décima iteración.

Tarea	Trabajo (horas
Revisión del código fuente de ambos desarrollos	4
Revisión de los resultados	3
Revisión de la documentación	4
Total cierre de proyecto	**11**

Tabla 2.4: Planificación del cierre de proyecto

3

Desarrollo de la plataforma

En este capítulo se determinarán los requisitos del sistema, cómo se compone, y los diagramas de la arquitectura.

3.1. Determinación del alcance del sistema

El sistema a implementar presenta dos ramas de desarrollo que pueden y deben interactuar entre sí: Un servicio web creado con Flask [43] y una aplicación Android. Los servicios web están diseñados como una API REST que permite interactuar con el sistema mediante peticiones HTTP. También se proporciona un *frontend* básico que permite interactuar con el sistema de manera más cómoda y sencilla para el usuario. La aplicación de Android, presenta una interfaz que sigue los principios de usabilidad y accesibilidad, y permite realizar las operativas que los servicios web de forma nativa, sin una API.

Ambos sistemas incluyen operativas como poder calcular PSI, generar claves nuevas, añadir *peers* en redes de área local o generar nuevos conjuntos secretos. Los sistemas dependen de la concurrencia en el sentido de recepción de mensajes y ejecución de operaciones, se implementan ejecutores prioritarios para hacer una gestión de la memoria lo más eficiente posible para los dispositivos. Los sistemas son modulares, posibilitando la extensión a otras técnicas o variantes de PSI, las operaciones que se necesitan realizar por parte de los distintos criptosistemas, así como los criptosistemas en sí.

El objetivo del sistema de Python es ofrecer una demostración de lo que se puede realizar, por lo que, mientras que el sistema es concurrente, no es multiproceso, ni se ha considerado realizarlo así. Esto quiere decir que está limitado por el *Global interepter Lock* (GIL) de Python [44], por ende, el interpretador solo podrá ser utilizado por un hilo a la vez, los demás tienen que esperar a poder adquirir el GIL, bien porque el hilo que lo tuviera esté esperando, porque se haya quedado ocioso, o porque se le haya obligado a liberarlo.

3.2. Análisis

3.2.1. Requisitos del sistema

Se presentarán requisitos funcionales comunes, presentes en ambos desarrollos, y requisitos específicos, estos quedan denotados por *RFA-X* y *RFWS-X*, respectivamente, donde la X representa el número de requisito. Los requisitos no funcionales se definirán como *RNF-X*. Se detallan en el Apéndice A.2.

Compartidos

RF-1. El sistema debe permitir crear un nodo.

RF-1.1. Esto debe realizarse por defecto, e incluir una opción para hacerlo manualmente.

RF-1.2. Solo un nodo corriendo en el mismo dispositivo de forma simultánea.

RF-2. El sistema debe permitir destruir el nodo activo.

RF-3. El sistema permitirá conectarse a otros compañeros de la red.

RF-3.1. El nodo debe estar inicializado.

RF-3.2. Podrán ser servicios web o dispositivos Android.

RF-3.3. Todos los compañeros deben estar corriendo el mismo puerto abierto.

RF-3.4. La conexión debe ser no discriminatoria, si la IP y el puerto configurados son accesibles, se debe poder conectar.

RF-3.5. Se podrá conectar utilizando una función de descubrimiento si el dispositivo está en una red de área local.

RF-4. El sistema debe permitir hacer ping para comprobar la conexión.

RF-4.1. En caso de recibir respuesta a ping, el dispositivo debe actualizar su última hora de conexión. Se debe indicar en caso de que no se reciba respuesta.

RF-4.2. Se podrá realizar independientemente de su última hora de detección.

RF-5. El sistema debe permitir iniciar procesos de intersección con un compañero.

RF-5.1. Con cualquiera de las técnicas y esquemas criptográficos configurados.

RF-5.2. Independientemente de su hora de última detección.

RF-6. El sistema debe permitir lanzar *tests* contra un compañero.

RF-6.1. Las operaciones, criptosistemas y técnicas pueden ser personalizados.

RF-6.2. Se podrá usar una opción por defecto.

RF-7. El sistema permitirá generar nuevas parejas de claves públicas-privadas.
RF-7.1. Se debe seleccionar el criptosistema para generar nuevas claves.
RF-8. La aplicación debe permitir visualizar los resultados de intersecciones obtenidos junto al tipo y el resultado de ésta.
RF-9. El sistema debe permitir obtener/visualizar el conjunto de datos actual.
RF-10. El sistema debe permitir la obtención/visualización de claves públicas para cada criptosistema.
RF-11. El sistema debe permitir parar el envío de registros a la Realtime Database de Firebase.

Específicos Android
RFA-1. Se podrá conectar introduciendo la dirección de un compañero.
RFA-2. La aplicación debe mostrar el estado de la red en todo momento.
RFA-3. La aplicación debe mostrar la lista de dispositivos descubiertos o registrados en su red.
RFA-3.1. Los dispositivos mostrarán la última hora que interactuaron con el dispositivo.
RFA-4. La aplicación debe permitir modificar el tamaño y dominio de su conjunto de datos.
RFA-5. La aplicación debe notificar al usuario cada vez que se completa una acción.
RFA-5.1. Cuando se generan claves.
RFA-5.2. Cuando se completa cualquier paso de la intersección.
RFA-6. La aplicación debe actualizar la lista de dispositivos si se encuentra uno nuevo, o un desconocido se comunica con él.

Específicos Servicios web
RFWS-1. El servicio debe permitir hacer llamadas a la API REST desde el *frontend* proporcionado.
RFWS-2. La API debe devolver los dispositivos de la red registrados para el nodo cuando se le solicite.
RFWS-2.1. Esto incluye sus identificadores y fechas de última vez visto.

3.2.2. Identificación de los actores del sistema

Los actores que pueden interactuar con el sistema son *Usuario anónimo*: Usuario con conocimientos de informática suficientes, siendo recomendable que sea capaz de interactuar con una API REST mediante peticiones HTTP. Puede ser un *"Usuario conectado/Compañero"*, con acceso a todas las funcionalidades y susceptible de ser detectado o añadido por otro Usuario, o un *"Usuario desconectado*, dependiendo de si su instancia de nodo está activa. Solo puede desactivar el envío de registros e e iniciar el nodo en un puerto de su elección.

Por defecto se instancia un nodo cuando se inicia el sistema por primera vez,

y la función de conexión/desconexión está pensada para dejar de estar conectado a la red, para escuchar en un puerto específico, o para conseguir una instancia de nodo nueva (nuevos datos, nuevos criptosistemas, sin compañeros conocidos).

Los principales casos de uso se muestran en la Figura 3.1 y después se detalla cada uno de ellos en el Apéndice A.3.

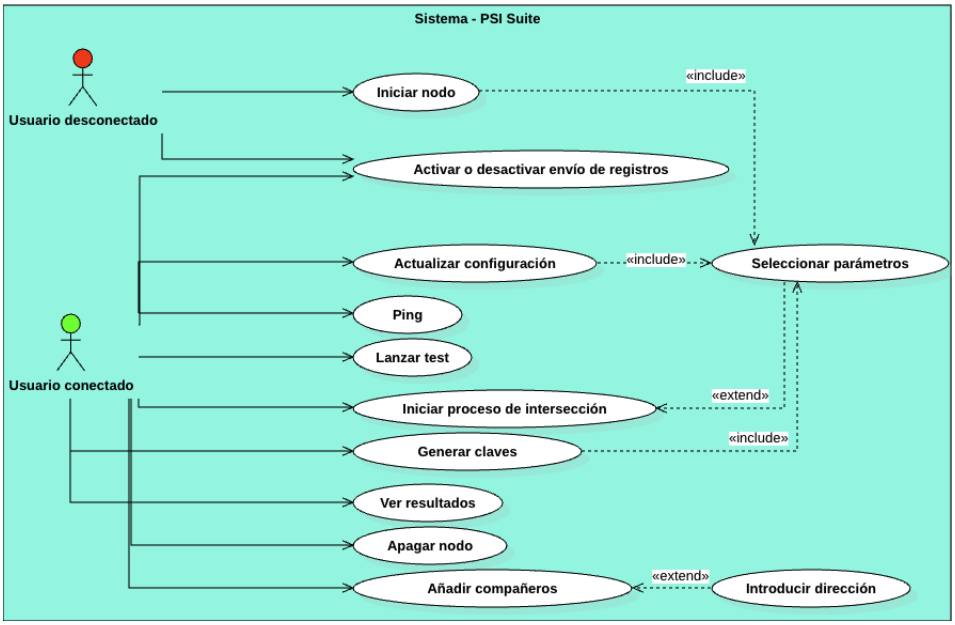

Figura 3.1: Diagrama de casos de uso

3.3. Diseño del sistema

Para diseñar el sistema, además de utilizar la investigación realizada anteriormente, se han tenido que probar varias tecnologías y ver cómo escalaban, para asegurar que ofrecieran unas soluciones flexibles y escalables. Por ejemplo, se han estudiado las distintas alternativas para poder crear la red descentralizada entre los servicios web de Python y Android.

3.3.1. Arquitectura

Diagrama de paquetes

En la Figura 3.2 se detallan los principales paquetes que contiene cada sistema, separados por el sistema Android y el sistema Python. La separación en paquetes de Android se ha llevado a cabo utilizando la metodología que se suele utilizar para separar lógica de negocio de actividades y servicios, mientras que en Python

se ha utilizado una separación basada en las responsabilidades de cada uno de los paquetes.

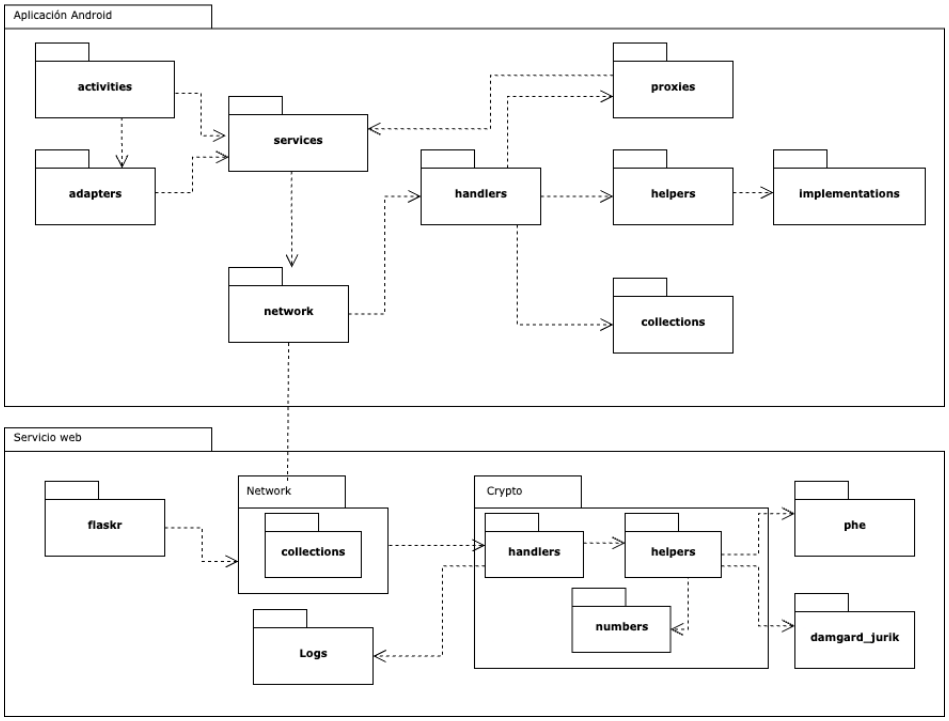

Figura 3.2: Diagrama de paquetes

Android

activities: El paquete que contiene las distintas actividades de la aplicación. En este caso la principal y la que se utiliza para mostrar distinta información.

Adapters: Incluye el adaptador de dispositivos que utiliza la aplicación. para poder visualizarlos en la actividad principal.

Services: Los servicios de Android que necesita iniciar la aplicación para funcionar. El servicio de red, encargado de mantener y poseer el nodo y actuar como fachada entre la interfaz y éste, con el objetivo de mantener la fiabilidad del nodo al poder seguir corriendo en segundo plano, y para separar la lógica de negocio de las actividades; y el servicio de registros, con la lógica necesaria para la autenticación y el registro de operaciones a la base de datos.

network: Paquete responsable de la red descentralizada, esto implica toda la lógica que permite el envío y recepción de mensajes, así como la asignación de prioridad de operaciones y la gestión de mensajes JSON que se envían o reciben. En este paquete se incluye el objeto que permite asignar prioridad a las operaciones que realiza el nodo, el objeto que representa un dispositivo en la red, el gestor de archivos JSON y el nodo en sí.

handlers: Este paquete incluye la lógica de los distintos protocolos de PSI, se llama de este modo porque realizan lo que sería la gestión de cada protocolo de principio a fin, dependiendo del paso que le mande el gestor de JSON.

helpers: Incluyen la capa de lógica o cobertura que se le da a los criptosistemas

en base a las operaciones que se necesita de ellos (*Facades*), lo cual se especifica en una interfaz, con el objetivo de reducir las llamadas al objeto del criptosistema en sí y facilitando el trabajo de los gestores de protocolos, facilitando la integración con posibles librerías externas, al poder dar cobertura a nuevos criptosistemas sin tener que conocer cómo funcionan.

implementations: El paquete encargado de los dos criptosistemas implementados que cumplen con la misma interfaz.

collections: Paquete que incluye colecciones estáticas de variables y la colección de métodos estáticos de polinomios. También incluye los métodos necesarios para saber el tamaño de los objetos en memoria y el enumerado con los distintos nombres que un criptosistema puede tener (para hacerlo más compatible con otros nombres o implementaciones, y por conveniencia).

proxies: Paquete que representa una parte del sistema de registros, encargados de implementar el patrón *proxy* como se puede comprobar a continuación en el diseño de clases. Encargado de actuar como una capa entre quien solicita empezar a guardar métricas y el servicio de registros, pudiendo añadir operaciones en un intermediario, en este caso solo hay trazas de depuración ADB, pero se podría expandir esta capacidad.

Python

flaskr: Incluye el archivo Python de inicialización del sistema que se encarga de exponer la API REST y de enlazar las plantillas HTML de la interfaz gráfica. Los archivos JavaScript de este directorio definen funciones que sirven para interactuar con la interfaz gráfica. Toda interacción con el sistema subyacente se realiza desde aquí, abstrayendo toda la lógica de cara al usuario.

Network: Como en Android, es el paquete responsable de la gestión del nodo de la red descentralizada. Se incluye el nodo en sí, con su configuración y métodos correspondientes para almacenar compañeros, escuchar y enviar mensajes utilizando ZMQ. Este paquete incluye también el gestor de archivos JSON, que también van a salir o entrar mediante la red, y la configuración del ejecutor de prioridad, responsable de asignar una prioridad u otra a las distintas operaciones que realiza el sistema.

Hay un paquete dentro de *Network*, **collections**, que incluye variables estáticas que utiliza el sistema y métodos para la extracción y validación de direcciones IP.

Logs: Paquete que gestiona todo lo relacionado con registros, esto incluye el objeto único donde cada hilo almacenará su información, así como la autenticación de Firebase y la lógica para la creación y envío de los registros a la base de datos.

Crypto: Paquete que incluye las envolturas de los criptosistemas o ***helpers***, capas de lógica que extraen de las librerías las operaciones necesarias para cumplir con la interfaz con la que funcionan los protocolos de PSI implementados, estos están en otro subpaquete ***handlers*** donde se configura la lógica de los distintos protocolos. Este paquete incluye la teoría de números utilizada (***numbers***) que se encarga de manejar las operaciones relacionadas con polinomios.

phe y **damgard_jurik**: Las librerías utilizadas como criptosistema base en este desarrollo para Paillier y Damgård-Jurik. En el código se incluye también ***py-fhe***, la librería utilizada y modificada como implementación del criptosistema de Brakerski-Fan-Vercauteren. No se incluye en este diagrama porque se ha integrado a modo de prueba de concepto, y no se ha implementado en Android.

Diagrama de despliegue

Cada aplicación es independiente, pero el propósito es que los dispositivos se conecten entre sí, como detalla el diagrama de despliegue de la Figura 3.3. Cuando el sistema se inicia, por defecto se levanta un nodo para escucha y añade a los compañeros que se hayan preseleccionado (en el código fuente, si los hay), pudiendo apagarlo para levantar otro nodo que escuche en otro puerto distinto al original.

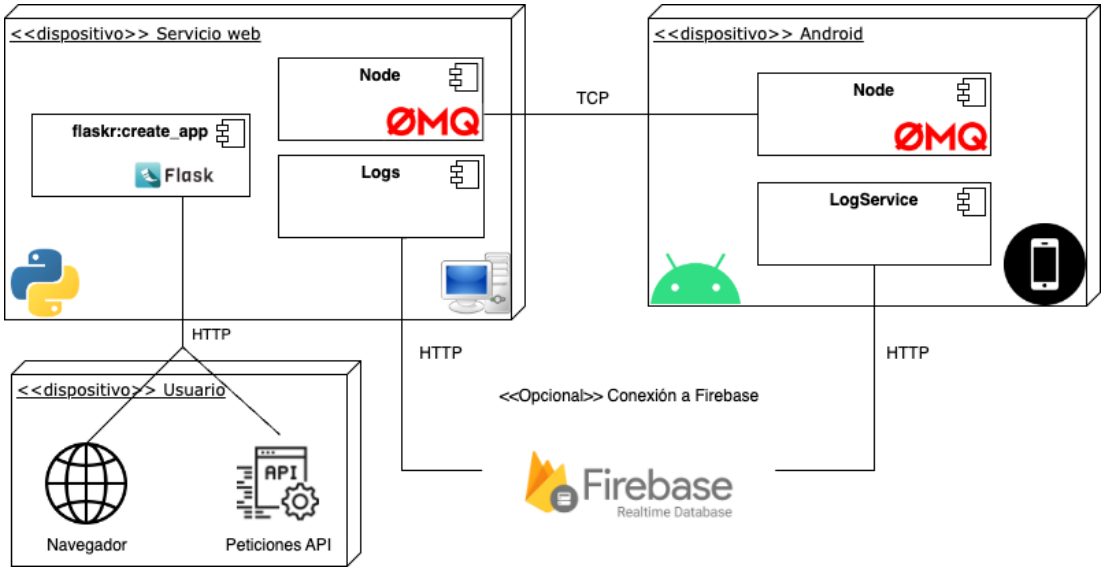

Figura 3.3: Diagrama de despliegue

La comunicación entre dispositivos ser realiza usando sockets de ZeroMQ (ZMQ), que es una biblioteca de mensajería asíncrona de alto rendimiento[45] que en este caso se utiliza como base para la red descentralizada. Se utilizan sockets *DEALER*, teniendo uno por dispositivo, utilizados para enviar mensajes, las direcciones quedan vinculadas a un destino con el que se comunicará mediante TCP. Para recibir mensajes, se utiliza el socket *ROUTER*, que queda vinculado a la interfaz de red con la dirección de red de área local del dispositivo. Los detalles de esta arquitectura se comentan más adelante, en la Sección 3.3.6. Las aplicaciones, si disponen de la autenticación necesaria, se conectan también a la Firebase Realtime Database del proyecto asociado para enviar los registros, esto es opcio-

nal, no hay obligación de tener un proyecto de Firebase activo y autenticado para utilizar el sistema, pero se perderá la capacidad de almacenar registros.

Se puede interactuar con el servicio web de Flask desde un navegador accediendo al *frontend* desarrollado, o mediante peticiones HTTP a la API REST.

En el código proporcionado, existe la posibilidad de crear una imagen de Docker[46] con el archivo *Dockerfile* y *dockerstart.sh* necesarios para ello y se proporciona un archivo *docker_compose.yml* que levantará cuatro servicios web independientes bajo la misma red de Docker, pudiéndose conectar entre sí. Se planteó esta posibilidad con el objetivo de ofrecer un servicio que se pudiera levantar desde una imagen y crear redes conectadas fácilmente. No se ha utilizado porque la recolección de métricas no funcionaba y las limitaciones de tiempo no permitieron llevar a cabo un diagnóstico para solucionarlo. No obstante, se ofrece esta forma de levantar servicios y se ha probado su funcionamiento.

3.3.2. Diseño de clases

A continuación, se presenta el diagrama final de las clases de ambos sistemas. Este incluye cambios que se hicieron desde el principio, y representa la estructura del sistema entregado. Para facilitar la legibilidad, se han omitido detalles como *getters* y *setters*, y se han separado las actividades y servicios de la arquitectura de la red y la criptografía en el caso de Android.

El diagrama de clases de red y criptografía de Android se puede ver en la siguiente página, en la Figura 3.4.

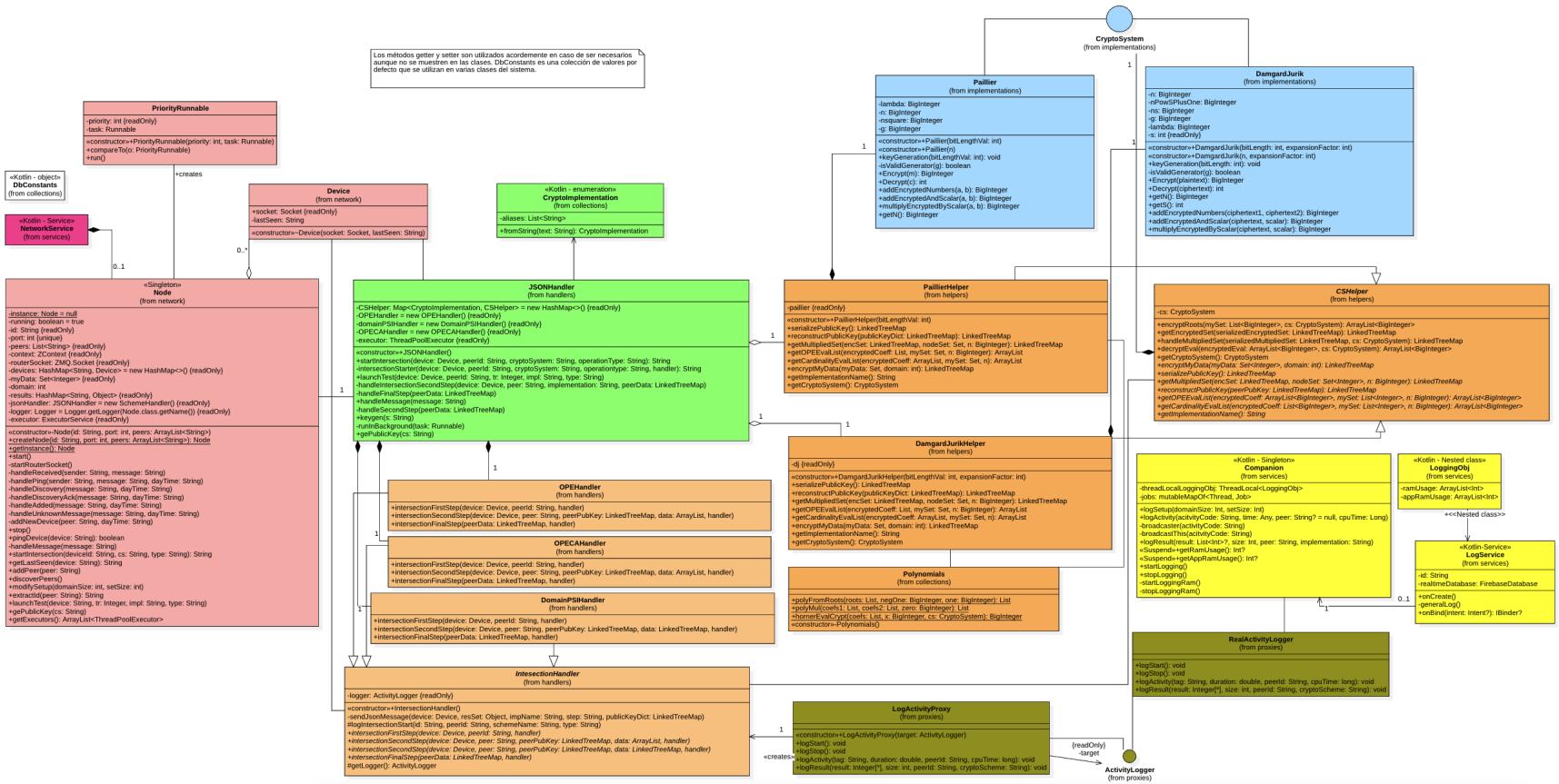

Figura 3.4: Diagrama de clases de red y criptografía - Android

Este diagrama para el desarrollo Python, muy similar al de Android, se puede encontrar en la figura del Apéndice A.4.

Descripción de las clases

Los sistemas son muy similares, intentando que sea extensible a nuevas técnicas o variantes para calcular PSI, criptosistemas, u operaciones. A continuación, se describen las clases más relevantes del sistema.

Node - Singleton

Descripción

Solo puede haber un nodo en el sistema. Este nodo escucha mensajes en el socket *ROUTER* de ZeroMQ (en el puerto establecido) y los consume, delegando su ejecución a otras clases cuando sea conveniente. Además, permite a las interfaces interactuar con él para conseguir el resutlado que quieren, escondiendo las complejidades específicas de cada operación. Se usa una especie de patrón *Command* para manejar de mensajes recibidos de forma adecuada dependiendo de su contenido.

Atributos

Identificador, puerto, contexto ZeroMQ, socket *ROUTER*, compañeros, JSONHandler, datos (secretos), dominio en el que opera, resultados y ejecutor de prioridad.

Métodos

Acciones para consumir datos recibidos

startRouterSocket: Crea un hilo en el que el nodo empezará a escuchar por mensajes hasta que se pare su ejecución.
handle: Diferentes métodos para consumir apropiadamente el mensaje recibido (*Received, Ping, Discovery, DiscoveryAck, Added, UnknownMessage, Message*)
handleMessage: Delega en *JSONHandler* ya que el mensaje recibido es un JSON.

Acciones para dar soporte a las interfaces

stop: Cierra los sockets y destruye la instancia del nodo.
pingDevice: Manda un ping a un compañero conocido.
startIntersection: Delega la responsabilidad en el *JSONHandler* para que encole la operación en su ejecutor y compruebe si los parámetros (método y criptosistema) son apropiados, este a su vez la delegará en el *IntersectionHandler* apropiado.
discoverPeers: Manda un mensaje de conexión en las direcciones conocidas de una red de área local, quien le responda, será añadido a la red.
modifySetup: Actualiza los "secretos"del nodo.
launchTest: Delega en *JSONHandler*.
El resto son *getters* y funciones de ayuda para las interfaces.

Device

Descripción

Objeto que incluye los detalles de otro dispositivo de la red.

Atributos

socket: El socekt *DEALER* de ZMQ que se utilizará para comunicarse con el dispositivo.
lastSeen: Indica la fecha y hora en la que se interactuó con dicho dispositivo.

JSONHandler

Descripción

Actúa como un intermediario con la lógica necesaria para selección de métodos y criptosistemas que se utiliza con todas las operaciones que van a serializarse, o que vienen serializadas. Al incluir el criptosistema, se encarga también de la generación de claves

Atributos

CSHelper: Un HashMap que facilita el acceso a un criptosistema específico.
Hanlders específicos: Para cada uno de los protocolos utilizados para calcular PSI.
executor: un *ThreadPoolExecutor* que se iniciará con una cola de prioridad y al que se le añadirán funciones para ejecutar con prioridades.

Métodos

runInBackground: Crea un objeto de tipo *PriorityRunnable* y lo sube al ejecutor, sin bloquear la ejecución del programa principal.
handleMessage: Se encarga de tratar el mensaje como JSON y enviarlo al método apropiado dentro de la clase.
startIntersection: Le encola una operación al ejecutor indicando el criptosistema y se lo manda a *intersectionStarter*
intersectionStarter: Decide cuál es el protocolo apropiado y le manda la acción al protocolo adecuado.
launchTest: Encola las operaciones al ejecutor de las iteraciones y el método que se deba utilizar.
handleSecondStep: Se encarga de coger los datos relevantes del JSON y aplicar la lógica necesaria para encolar la tarea correspondiente al ejecutor.
handleFinalStep: Igual que el anterior pero para mensajes que vienen con el indicador de paso final.
keygen: Un hilo genera las nuevas claves para el criptosistema seleccionado con la longitud de claves seleccionada.

IntersectionHandler

Descripción

Es una clase abstracta, sus implementaciones concretas se encargan de realizar las operaciones necesarias para cumplir cada uno de los pasos necesarios para calcular una intersección.

Atributos

logger: Un objeto de tipo ActivityLogger al que se llamará para que realice las operaciones relacionadas con el registro de datos.

Métodos

sendJsonMessage: Envía un mensaje usando el socket del objeto *Device*. Como estos mensajes son únicos para las intersecciones, se han quedado aquí, cualquier otra operación con JSON podría tener que enviar mensajes distintos a estos.

intersectionFirstStep (Abstracto): Realiza el primer paso de la intersección, dependerá de la implementación específica.
intersectionSecondStep (Abstracto): Realiza el segundo paso de un proceso de cálculo de intersección con un mensaje que recibe. Dependerá de la implementación concreta, que será la que determine el *JSONHandler.* en base al mensaje recibido que le pase el nodo.
intersectionFinalStep (Abstracto): Como los dos anteriores, dependerá de una implementación concreta y su ejecución dependerá de lo que decida hacer el *JSONHandler.*

CSHelper
Descripción
Clase abstracta, sus implementaciones concretas implementan métodos de ayuda para trabajar con conjuntos o listas de datos. Es una interfaz o fachada para interactuar con el criptosistema de manera sencilla, sin tener que utilizar los métodos primitivos de este. Ayuda con temas de serialización, deserialización, y trabajos concretos sobre datos cifrados.
Atributos
cs: Criptosistema con el que se está trabajando.
Métodos
encryptoRoots: Método que cifra las raíces de un polinomio previamente calculadas.
getEncryptedSet: Deserializa un conjunto de datos recibido.
handleMultipliedSet: Descifra los datos de un conjunto recibido.
decryptEval: Descifra los datos de una lista recibida.
encryptMyData: Cifra el conjunto de datos de la parte que solicita un inicio de cálculo de intersección.
serializePublicKey: Serializa la clave pública.
getMultipliedSet: Realiza el paso 2 de la intersección por dominio.
reconstructPublicKey: Reconstruye una clave pública que ha venido serializada.
getOPEEvalList: Obtener el resultado de las evaluaciones para el paso dos del protocolo Private Matching.
getCardinalityEvalList: Obtener el resultado de las evaluaciones para el paso dos del protcolo Private Matching para la cardinalidad.

CryptoSystem - Interfaz - Android
Descripción
Las clases que implementan esta interfaz incluyen las implementaciones de los criptosistemas. Las implementaciones concretas contienen detalles de las claves públicas y privadas y están diseñadas para poder realizar las operaciones más básicas de un criptosistema homomórfico.
Atributos (Implementaciones concretas)
Los parámetros necesarios para tener la clave pública y privada. Se pueden añadir precomputaciones para mejorar el rendimiento en algunas operaciones.
Métodos

keyGeneration: Genera una pareja de claves públicas y privadas dependiendo del criptosistema en el que se ejecute.
Encrypt: Cifra un *BigInteger* dado usando el esquema que lo implemente.
Decrypt: Descifra un BigInteger dado usando el esquema que lo implemente.
multiplyEncryptedByScalar: Multiplica un número cifrado por un escalar usando las propiedades homomórficas.
addEncryptedNumbers: Suma dos números cifrados usando las propiedades homomórficas.
addEncryptedAndScalar: Suma un número cifrado y un escalar usando las propiedades homomórficas.

Aclaración

Esta interfaz proporciona soporte, en principio, para esquemas criptográficos parcialmente homomórfico (PHE) aditivos que permitan realizar las tres operaciones homomórficas propuestas, es extensible a otros esquemas.

ThreadData / LoggingObj

Descripción

Un objeto local a cada hilo con las mediciones de cada operación medible. En Android, se usa la capacidad de tener un objeto local para cada hilo y en Python se pasa el objeto cada vez que se quieren realizar lecturas o escrituras sobre él. Los objetos se destruyen y son eliminados por el recolector de basura periódicamente.

Atributos

Lo que se puede medir en cada plataforma, en Python tendrá lecturas de CPU y RAM, en Android, cómo no hay manera de poder leer el uso de la CPU por motivos de seguridad [47], se pasará el uso de RAM de la máquina virtual, el uso de RAM del dispositivo y el tiempo de uso de la CPU.

La Figura 3.5 de la siguiente página contiene el diagrama de clases de Kotlin utilizadas para las actividades y servicios de Android. La actividad ***MainActivity*** levanta los servicios necesarios: ***NetworkService***, que se encargará de todas las acciones relacionadas con la red, es decir, será el punto de acceso de la actividad a la gestión del nodo y actúa como una fachada para los métodos complejos, este se mantendrá activo en segundo plano; y ***LogService***, que se encargará de autenticar al usuario, de registrar los consumos de las operaciones, y de realizar las escrituras pertinentes a la base de datos.

MainActivity incluye opciones interactuar con los servicios y mostrará en pantalla los compañeros recuperados del nodo activo usando el adaptador ***DeviceListAdapter***, que permite realizar acciones contra un dispositivo concreto de forma simple y visual. Puede crear actividades ***KeysActivity***, utilizada como plantilla para mostrar los datos que el usuario quiera ver como resultados, claves o su conjunto de secretos.

Figura 3.5: Diagrama de clases de la interfaz - Android

3.3.3. Diagramas de interacción

Para este apartado, se va a detallar el caso de uso más importante **CU-005 - Iniciar cálculo de PSI**, se mostrará en detalle el funcionamiento de

la operativa desde un dispositivo Android a un servicio web y cómo actúa este último cuando recibe el mensaje. En este caso se simulará lo que sería el cálculo de una intersección privada usando el protocolo Private Matching con evaluaciones polinómicas.

Un usuario conectado utilizando la aplicación Android tiene el nodo activo y un servicio web conectado en su lista de dispostivos conocidos. El usuario selecciona la técnica y criptosistema a utilizar. La Figura 3.6 detalla el primer y segundo paso del cálculo de PSI, el cifrado de datos y la evaluación de estos datos por la parte destinataria. En el segundo paso, el servicio web recibe el mensaje del dispositivo que está intentando calcular la intersección con él, y realiza los cálculos necesarios según los parámetros del mensaje recibido. Los hilos que realizan cada acción están marcados con distintos colores.

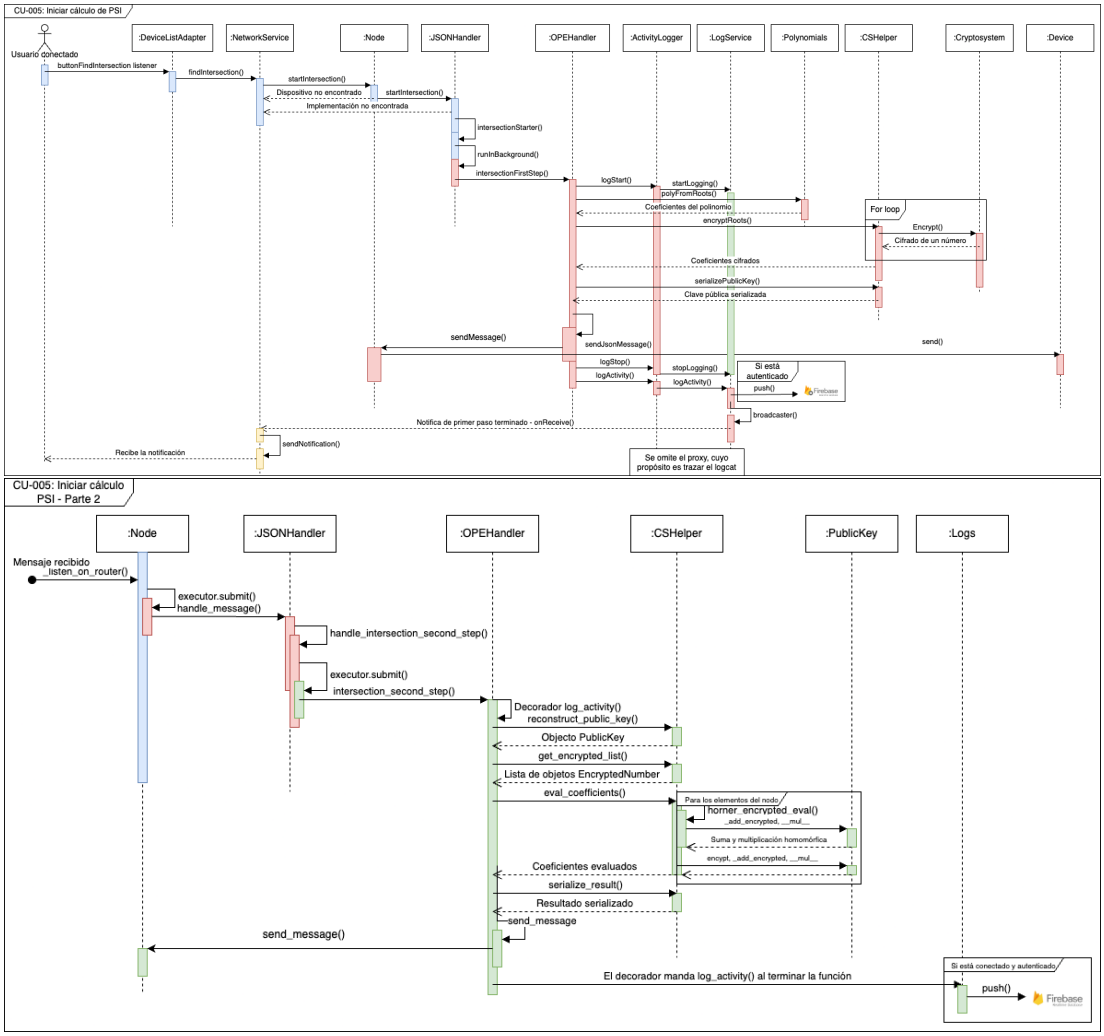

Figura 3.6: Diagramas de interacción - Cifrado y envío de datos en Android (arriba), Evaluación de datos recibidos en servicio web (abajo)

3.3.4. Plan de pruebas

Aquí se describirán las pruebas unitarias, pruebas de integración, regresión y manuales propuestas para validar el funcionamiento del sistema.

Pruebas unitarias

Para la realización de estas pruebas, se ha utilizado el *framework* unittest [48] en Python, y JUnit 4 [49] en Android. El objetivo de estas pruebas es validar el funcionamiento de las partes individuales del sistema, como la gestión de dispositivos en el nodo, los criptosistemas, o el cálculo de intersecciones con una técnica específica. Se han propuesto las siguientes pruebas para cada línea de desarrollo:

- Nodo:
  - Creación de nodos y verificación de su estado inicial.
  - Comunicación simulada entre nodos.
  - Gestión y validación de direcciones IP.
  - Adición y eliminación de compañeros.
- Criptosistemas (solo Android):
  - Generación y almacenamiento de claves.
  - Operaciones de cifrado y descifrado.
  - Verificación de operaciones homomórficas.
- Protocolos PSI:
  - Simulación de cálculos de intersección.
  - Pruebas de operaciones polinómicas utilizadas en PSI.

Pruebas manuales y/o exploratorias

Estas pruebas no siguen un patrón común, se basan en explorar el sistema con el objetivo encontrar fallos o áreas de mejora. Esto incluye el buscar casos límite o encontrar casos que no se hayan contemplado en el resto de pruebas. Ejemplos de descubrimientos gracias a estas pruebas incluyen fallos en la API REST de Python relacionados con la falta de verificación de entradas, problemas visuales en la interfaz de usuario y la gestión inestable de la pausa y reanudación del servicio de red en Android.

Estos fallos se corrigieron y no se habrían encontrado sin estas pruebas, están detallados en la Sección 3.3.7.

Pruebas de integración

Las pruebas de integración han hecho que se pueda validar el funcionamiento de todo el sistema como un conjunto, desde la comunicación más básica entre dispositivos hasta el registro de métricas en la base de datos. Los flujos que se han comprobado en estas pruebas incluyen:

- Verificación de la comunicación básica entre dispositivos heterogéneos y del mismo tipo utilizando ZMQ.
- Validación del flujo completo de datos desde el envío hasta la recepción de un mensaje básico (*ping, descubrimiento...*).
- Validación del flujo completo de PSI entre dispositivos heterogéneos y del mismo tipo con distintos esquemas y protocolos.
- Validación del flujo de registro de datos en la base de datos y verificación de datos subidos.

Pruebas de regresión

El objetivo de estas pruebas es comprobar que al añadir una nueva funcionalidad el resto del sistema sigue funcionando como debe. Aunque la modularidad del sistema minimiza los errores al agregar características, estas comprobaciones son necesarias en cualquier desarrollo. Incluyen:

- Validación de que los componentes existentes funcionan correctamente tras la integración de nuevos criptosistemas o protocolos.
- Ejecución de pruebas unitarias y pruebas de integración para todos los componentes relevantes.
- Comparación del rendimiento antes y después de la integración para identificar posibles degradaciones.

Varias comprobaciones se realizan en la etapa de pruebas unitarias y pruebas de integración, yo como desarrollador he preferido considerarlas una extensión de las pruebas manuales, enfocándome en probar primitivas que ya funcionaban antes de validar el sistema completo.

3.3.5. Tecnologías

A continuación, se exponen las dependencias y herramientas utilizadas en el desarrollo del proyecto.

Dependencias

Las versiones específicas de las herramientas aquí presentes se encuentran definidas en los requisitos no funcionales.

- **Flask** [43]: Microframework para Python empleado en la creación de servicios web y API REST.
- **Waitress** [50]: Servidor WSGI utilizado para desplegar los servicios web de Flask de forma que simule un entorno productivo.
- **phe** [37]: Librería utilizada como esquema criptográfico de Paillier para el desarrollo de Python.
- **damgard-jurik** [38]: Librería utilizada como esquema criptográfico de Damgård-Jurik en el desarrollo de Python.
- **py-fhe** [41]: Librería utilizada como esquema criptográfico BFV en el desarrollo de Python. Es un añadido como prueba de concepto de un criptosistema de este calibre.
- **ZeroMQ** [45]: Biblioteca de mensajería asíncrona utilizada para el envío de mensajes entre dispositivos en la red descentralizada.
- **Firebase** [51]
  - **Firebase Realtime Database** [52]: Base de datos en tiempo real utilizada para registrar los recursos consumidos, tiempos de operación y otros datos relevantes.
  - **Crashlytics** [53]: Herramienta de informes de fallos utilizada en la aplicación Android para monitorizar y solucionar problemas de manera eficiente sin necesidad de estar depurando mediante ADB.

El archivo *build.gradle.kts* en el desarrollo de Android incluye otras dependencias propias de Android y necesarias para su funcionamiento, añadidas de manera automática por el IDE. Algunas añadidas manualmente incluyen *Mockito* [54] para el desarrollo de algunas pruebas unitarias o las de *Firebase*, necesarias, por ejemplo, para el uso de la autenticación. Python incluye sus dependencias necesarias para funcionar en el archivo *requirements.txt*, esta incluye también a *psutil* [55], necesaria para poder medir el uso de recursos de la máquina que está corriendo el servicio.

Herramientas y programas usados en el desarrollo

- **Android Studio** [56]: IDE o entorno de desarrollo de Google en colaboración con JetBrains enfocado a los lenguajes Java y Kotlin para el desarrollo

de aplicaciones Android, facilitando la integración con herramientas como Firebase, y proporcionando herramientas como la consola ADB o emuladores de dispositivos.

- **Pycharm** [57]: IDE o entorno de desarrollo de JetBrains enfocado en el lenguaje Python con herramientas específicas como el gestor de entornos o la consola de Python integrada. Utilizado para el desarrollo de los servicios web.
- **Postman** [58]: Herramienta utilizada para probar y documentar la API REST expuesta por el servicio web. Permite realizar peticiones HTTP y verificar las respuestas del servidor.
- **Git** [59]: El software de control de versiones, en este proyecto se ha utilizado para controlar el desarrollo en dos repositorios independientes (servicios web y aplicación Android).
- **Docker**[46]: Utilizado para levantar múltiples servicios web en una misma red en forma de máquinas virtuales independientes y en un mismo sistema y comprobar su funcionamiento y rendimiento con múltiples dispositivos.
- **GitHub Desktop** [60]: Herramienta desarrollada por GitHub para facilitar el control de versiones, utilizada principalmente para la gestión sencilla de las ramas.
- **StarUML** [61]: Herramienta utilizada para la creación de diagramas UML, facilitando el diseño y la documentación de la arquitectura del sistema.
- **draw.io** [62]: Herramienta que permite crear todo tipo de diagramas. Se ha utilizado la aplicación web para crear el diagrama de despliegue y el de paquetes.

3.3.6. Implementación

En los apartados inferiores se presenta la implementación de los componentes más complejos de ambas ramas de desarrollo.

Red descentralizada

Para establecer la comunicación entre dispositivos, teniendo en cuenta que estos son servicios web Python y dispositivos Android, sin necesidad de depender de un servidor, se valoraron distintas formas de conseguirlo. El proyecto necesita una solución que soporte mensajería asíncrona, escalabilidad y completamente distribuida. Se han evaluado sistemas de mensajería y opciones como marcos RPC y WebSockets.

1. gRPC [63]: Un marco de RPC desarrollado por Google. Es moderno y tiene buen rendimiento, pero está más limitado en temas de escalabilidad, en una red descentralizada no te puedes permitir tener esta limitación. Aunque soporta comunicación asíncrona, no es lo más adecuado, está recomendado como alternativas a REST u otros mecanismos que se usan para conectar clientes con servidores mediante APIs [64].

2. RabbitMQ [65]: Es un broker de mensajes que tiene soporte para la mensajería distribuida. El problema es el uso de recursos, teniendo en cuenta que había dispositivos Android, no era la opción más factible [66].

3. MQTT [67]: Es un protocolo de mensajería muy ligero que se utiliza mucho para comunicación en tiempo real entre dispositivos Android. El problema, tiene limitaciones con grandes conjuntos de datos, y los valores cifrados de los conjuntos contribuyen a tener que mandar archivos JSON bastante pesados [66]. No obstante, se podría haber utilizado y es la que habría seleccionado si en vez de dispositivos Android, se tratase de dispositivos IoT, con recursos muy limitados, dado que sus claves serían de menor tamaño, y por ende, textos cifrados también.

4. ZeroMQ [45]. A diferencia de las soluciones anteriores, ZMQ es una biblioteca de sockets de bajo nivel, con abstracciones para la comunicación asíncrona entre procesos, similar a cómo funcionan los sockets TCP/IP [68]. Al no tener un único punto de fallo como los brokers, la red puede escalar horizontalmente todo lo que quiera, y como es una biblioteca de bajo nivel, ofrece un alto rendimiento con menor consumo de recursos, lo que hace muy buena solución para Android [69]. Es una biblioteca muy bien documentada, lo que facilita la integración en el proyecto, aunque su desarrollo es más complejo que algunas alternativas.

5. Zigbee [70]. Un protocolo de comunicación diseñado para redes de dispositivos de bajo consumo y baja tasa de datos, pensado para IoT. Funciona bien en aplicaciones que requieren una transmisión de datos limitada y eficiente energéticamente, pero no es recomendable para aplicaciones que intercambien grandes volúmenes de datos o requieran alta velocidad y baja latencia. Su integración con dispositivos Android es compleja y menos directa comparada con otras tecnologías como MQTT o ZeroMQ. Zigbee puede crear redes de malla robustas y descentralizadas, pero está orientado para entornos de IoT. Su uso limitaría la aplicación a dispositivos compatibles con el protocolo, esto no pasa con el resto de alternativas.

Por la escalabilidad necesaria, queriendo una red descentralizada, y por el menor consumo de las soluciones anteriores, el proyecto se ha desarrollado usando ZeroMQ. A continuación se muestra cómo ha sido posible que los dispositivos heterogéneos se conecten entre sí y puedan intercambiar mensajes entre ellos, además de las ventajas de utilizar este enfoque.

Configuración inicial

Cuando se inicia el nodo, el sistema crea un socket *ROUTER* que se configura con el valor de la IP local del dispositivo y un puerto específico para escuchar en la red. Este socket *ROUTER* actúa como punto de entrada para los mensajes entrantes. Configurarlo con la IP local y un puerto específico asegura que el nodo esté disponible para recibir mensajes y solicitudes de otros nodos en la red.

Conexión entre dispositivos

La conexión entre dispositivos se realiza a través de mensajes de descubrimiento. Al iniciar, cada nodo crea un socket *DEALER* y envía un mensaje de *DISCOVERY* al dispositivo que quiere conocer. Los nodos receptores, al recibir un *DISCOVERY*, verifican si ya conocen al nodo remitente. Si no lo conocen, crean un socket *DEALER* para establecer una conexión directa con el nuevo nodo, respondiendo con un *DISCOVERY ACK*. Si ya lo conocían, le mandarán una "prueba de vida"para indicarle que ya le conocen y que el nodo solicitante cree un router *DEALER* permanente. Este mecanismo permite que los nodos se identifiquen y conecten entre sí fácilmente, añadiendo los nuevos nodos a su lista de dispositivos conocidos.

Esta arquitectura está basada en la guía oficial de ZeroMQ[1] en la que la combinación *DEALER* a *ROUTER* subraya a la perfección lo que se quiere, comunicación asíncrona entre servidores, que proporciona la flexibilidad necesaria para responder a los mensajes recibidos. También toma inspiración en literatura encontrada durante la investigación que plantea la posibilidad de un sistema así[2].

Manejo de mensajes

ZeroMQ actúa como una abstracción sobre sistemas de colas como MQTT o RabbitMQ[72], resultando en un modelo de comunicación basado en colas sin necesitar un servidor centralizado. En este sistema, cada nodo opera con un socket *ROUTER* que se encarga de recibir y encolar los mensajes entrantes. Cuando hay un mensaje pendiente en la cola del *ROUTER*, se desencola y se procesa según el tipo de mensaje. Este enfoque va más allá, priorizando los mensajes que requieren menor capacidad de cómputo como la respuesta a un *ping* o a un *DISCOVERY*, frente a mensajes que requieren más procesamiento como un JSON.

Ventajas

Tener un socket *ROUTER* en cada nodo permite que cada dispositivo escuche independientemente a otros nodos, facilitando la gestión de múltiples conexiones. Esto mejora el rendimiento, pues el *ROUTER* no tiene que ser consciente de los dispostivos que hay en la red, sino que consume los mensajes recibido, de esta forma, no tiene por qué saber si un dispositivo se ha desconectado. Esto va re-

[1] ZeroMQ Guide - Chapter 3 - The DEALER to ROUTER Combination, disponible en: https://zguide.zeromq.org/docs/chapter3/#The-DEALER-to-ROUTER-Combination

[2] Hintjens, P., "ZeroMQ: Messaging for Many Applications", 2013[71]

lacionado la tolerancia a fallos, si un dispositivo deja de funcionar, el resto de dispositivos solo nota que ha dejado de responder, pero pueden seguir funcionando entre ellos, lo mismo pasará si está fallando, simplemente descartarán sus mensajes.

De forma similar, tener un socket *DEALER* para cada dispositivo conocido en la red permite una comunicación directa y eficiente. Los sockets *DEALER* pueden enviar mensajes de forma independiente y no bloqueante, optimizando el envío y sin interrumpir el resto de la ejecución. Los socket *DEALER* permiten enviar mensajes específicos a nodos particulares, útil para tareas que requieren comunicación directa y específica entre nodos, en este proyecto, todas.

La flexibilidad y escalabilidad del sistema viene dada por ZeroMQ y por características de la arquitectura descentralizada que se ha desarrollado.

Flexibilidad:

- Autonomía de los nodos. Operan de forma independiente sin un servidor central y no necesitan reconfigurarse si un compañero abandona la red.
- Manejo de conexiones. Los dispositivos son capaces de conectarse entre sí conociendo su IP, cada uno se añadirá a la lista del otro y podrán empezar a comunicarse.
- Comunicación directa. Los sockets *DEALER* permiten que los nodos se comuniquen directamente entre sí una vez que se establece una conexión, mandar mensajes específicos de nodo a nodo se convierte en algo sencillo sin tener que pasar por intermediarios.

Esto hace que cada dispositivo o nodo, tenga la capacidad de adaptarse a cualquier cambio que haya en la red por sí mismos. El sistema proporciona una escalabilidad natural, la estructura permite añadir compañeros sin tener que cambiar la configuración existente. La red puede crecer de manera orgánica, incorporando nuevos nodos y conexiones cuando sea necesario. Pero no se queda ahí:

- Escalabilidad horizontal. Es prácticamente infinita añadiendo más nodos. Cada nodo se integra en la red con mensajes de descubrimiento y puede comenzar a operar en ella sin necesidad de volver a configurar la red.
- Distribución de la carga. Cada nodo hace lo que se le solicita, no se ponen a trabajar todos a la vez.
- Resistencia y recuperación. Si un nodo tenga problemas de conectividad no afectará al resto de la red, como se ha comentado, si se vuelve a conectar, podrá consumir los mensajes que se hayan quedado pendientes en los sockets de los emisores que le hayan intentado contactar, si no se han encolado más

de 2000 mensajes en los emisores, podrá consumirlos todos y no habrá perdido ninguno.

Una consideración a tener en cuenta es que manipulando múltiples sockets, es necesario fijarse en el consumo de memoria en los dispositivos Android, dado que esto podría degradar su rendimiento. Aunque los sockets no consumen mucha memoria, es necesario monitorizar los mensajes que están teniendo que consumir. En un entorno productivo con un número considerable de usuarios sería necesario tener herramientas de gestión de recursos como limitando el número de conexiones, por ejemplo. O similares a las de otros protocolos P2P como BitTorrent [73].

Concurrencia

Se ha diseñado un sistema concurrente para aprovechar las capacidades multihilo de Java, y en parte las de Python.

Escucha de mensajes
El socket *ROUTER* de ambos sistemas tiene un hilo ejecutándose siempre que el nodo esté activo, para consumir mensajes que llegue a la cola de ZMQ lo antes posible.

Sistema de registros
Para el sistema de registros, se lanzan hilos cuyo propósito es acceder a las métricas de uso del sistema, en Android, recogen el uso de RAM de la app (la que la máquina virtual de la aplicación tiene asignada), y el uso total de la RAM del sistema. Desde la API 26 no se puede acceder al uso de la CPU por motivos de seguridad [47]. En Python, se recogen las métricas de RAM y CPU usados por la instancia y por el sistema. Al final de la operación que se esté midiendo, si el sistema está autenticado, envía los datos recopilados a la Realtime Database de Firebase para ser analizados posteriormente.

Ejecutores prioritarios

En Android, al declarar el *ThreadPoolExecutor*, se establece que habrá un máximo de 10 hilos activos y que se utilizará la cola de prioridad bloqueante de Java, *PriorityBlockingQueue*, como cola de trabajo. Cuando se llame al método *execute* de este ejecutor, se encolarán objetos del tipo *PriorityRunnable* en la cola de trabajo, estos objetos contienen la tarea a ejecutar y su prioridad, de manera que la cola se encargará de que la siguiente operación a desencolar sea la de mayor prioridad.

La interfaz *Comparable* se implementa en *PriorityRunnable* para permitir la comparación de objetos *PriorityRunnable* entre sí. Si el ejecutor tiene hilos libres,

la tarea se asignará inmediatamente a uno de los hilos libres, sin necesidad de encolarla.

En Python no existe la posibilidad de cambiar directamente la cola del *ThreadPoolExecutor*, por lo que se ha creado la clase *PriorityExecutor*, que contiene el ejecutor real como atributo y una cola de prioridad, su función es ir desencolando tareas de la cola de prioridad y subirlas al ejecutor real. La tarea y la prioridad se envuelven en el objeto *PrioritizedItem*, la prioridad determina qué tarea sube antes al ejecutor.

La prioridad de ejecución del nodo procesa antes los mensajes de descubrimiento y *pings*, dejando con menos prioridad los mensajes JSON. Así, contesta cuanto antes a mensajes rápidos de otros nodos y que no piensen bien que el nodo no existe, o que no está respondiendo. Sería raro que se tuviera que aplicar esta prioridad, salvo que esté recibiendo muchos mensajes de seguido, el procesado que debe hacer es muy simple.

La prioridad de ejecución del *JSONHandler* se basa en eliminar las tareas recibidas para que los mensajes dejen de consumir memoria. Prioriza los mensajes de descifrado para obtener un resultado cuanto antes, luego van los mensajes que buscan una evaluación sobre un conjunto de datos cifrados, y, finalmente, las tareas de cifrado que envían el conjunto cifrado a un compañero (hasta que se ejecutan, no consumen memoria).

La razón de estas últimas prioridades es liberar memoria cuanto antes. Los mensajes cifrados ocupan mucho espacio en memoria, pudiendo llevar a problemas de lentitud o fallo del sistema en Android, donde la memoria es limitada por lo que pueda conceder el Tiempo de Ejecución de Android o *Android Runtime* (ART)[74] a la máquina virtual de la aplicación, hasta que llega un punto en el que no le va a dar más por limitaciones propias de Android. Si se quiere evitar esta limitación, se puede añadir *android:largeheap* [75] en el Manifest de Android para que la aplicación genere una pila de memoria más grande al abrirse, pero esto no es una solución apropiada y es mejor haber utilizado esta aproximación.

Esquemas de cifrado

Anteriormente, se han explicado las operaciones de cada criptosistema, para poder utilizar estas especificaciones, es necesario configurarlas apropiadamente en Java. A continuación, se muestra cómo se ha implementado el esquema de Paillier. El esquema de Damgård-Jurik es similar, cambiando el espacio de mensajes, se han implementado las operaciones homomórficas necesarias para los protocolos de PSI propuestos. La suma de un cifrado y un escalar no es necesaria pero se ha implementado como prueba de concepto.

Código 3.1: Generación de claves de Paillier en Java

```
public void keyGeneration(int bitLengthVal) {
        int certainty = 64;
        SecureRandom random = new SecureRandom();
        BigInteger p, q;
        do {
            p = new BigInteger(bitLengthVal / 2, certainty,
               random);
            q = new BigInteger(bitLengthVal / 2, certainty,
               random);
        } while (p.equals(q));

        n = p.multiply(q);
        nsquare = n.multiply(n);
        g = n.add(BigInteger.ONE);
        lambda = p.subtract(BigInteger.ONE).multiply(q.subtract
           (BigInteger.ONE)).divide(
                p.subtract(BigInteger.ONE).gcd(q.subtract(
                   BigInteger.ONE)));
        BigInteger a = g.modPow(lambda, nsquare).subtract(
           BigInteger.ONE).divide(n);
        if (!a.gcd(n).equals(BigInteger.ONE)) {
            throw new ArithmeticException("No se ha podido
               calcular mu, comprobar n y lambda.");
        } else {
            mu = a.modInverse(n);
        }
    }
```

Seguimos los pasos especificados, en este caso añadiendo un grado de certeza de que los primos generados p y q son primos.

Después, se calculan n y se precomputa n^2 ya que se utiliza en todas las operaciones. Se escoge g como $n+1$ por simplicidad aunque se comprueba que es un generador válido, esto se traduce a saber si g es coprimo con n y n divide el orden de g. Esta validación está codificada para entender más a fondo el criptosistema, aunque $g = n+1$ siempre será una opción válida porque $n+1$ es comprimo con n^2 para cualquier n al ser números consecutivos, en cualquier par de números consecutivos son coprimos.

Para terminar, aplicamos la fórmula para obtener λ y μ.

El resultado es que en el objeto Paillier habrá elementos de la clave pública (n, g) y de la privada (λ, μ).

Como está planteado, μ y g se podrían inferir para las operaciones a través de n y λ pero se precomputan como n^2 para acelerar los cálculos.

Código 3.2: Cifrado de Paillier en Java

```
public BigInteger Encrypt(BigInteger m) {
    SecureRandom r = new SecureRandom();
    BigInteger randNum;
    do {
        randNum = new BigInteger(n.bitLength(), r);
    } while (randNum.compareTo(n) >= 0 || !randNum.gcd(n).
       equals(BigInteger.ONE));
    return g.modPow(m, nsquare).multiply(randNum.modPow(n,
       nsquare)).mod(nsquare);
}
```

Como se explicó anteriormente, para el cifrado se genera un número aleatorio $r \in \mathbb{Z}_n^*$, es decir, un $r < n$, y se aplica la fórmula. Esta aproximación tiene la particularidad de que lo que estamos haciendo realmente es aplicar $c = (n+1)^m \cdot r^n \mod n^2$.

Es necesario que el número seleccionado sea coprimo con n y menor que n, esto se debe a que en el cifrado se utiliza r para elevarlo a la potencia de n mód n^2. Si r no es coprimo con n, $r^n \mod n^2$ no estará bien definido. Por otro lado, esto asegura que r tiene un inverso multiplicativo módulo n, necesario para el proceso de descifrado. Si el divisor común de n y r fuera mayor que uno, esto no se cumpliría, imposibilitando el proceso de descifrado

Código 3.3: Descifrado de Paillier en Java

```
public BigInteger Decrypt(BigInteger c) {
        return c.modPow(lambda, nsquare).subtract(BigInteger.
           ONE).divide(n).multiply(mu).mod(n);
    }
```

Los pasos originales se aplican directamente para calcular el mensaje original, utilizando la clave privada.

Aquí se ve explícitamente la necesidad de que el número aleatorio fuera coprimo con n, ya que utilizamos el inverso multiplicativo módulo n, y si no lo tuviera, se produciría una excepción que impediría el descifrado del número. Lo mismo pasaría si la n se hubiera generado de forma errónea y el mínimo común divisor de n no fuera uno.

Para la suma homomórfica de dos textos cifrados y las operaciones con escalares se puede ver en el Código 3.4 que la aplicación de la fórmula es directa.

Código 3.4: Operaciones sobre textos cifrados con Paillier en Java

```
public BigInteger addEncryptedNumbers(@NonNull BigInteger a,
   BigInteger b) {
        return a.multiply(b).mod(nsquare);
```

```
3       }
4   public BigInteger addEncryptedAndScalar(@NonNull BigInteger a,
        BigInteger b) {
5           return a.multiply(g.modPow(b, nsquare)).mod(nsquare);
6       }
7
8   public BigInteger multiplyEncryptedByScalar(@NonNull BigInteger
        a, BigInteger b) {
9           return a.modPow(b, nsquare);
10      }
```

La traducción anterior sigue lo citado en los estudios citados anteriormente.

No tiene por qué ser la mejor o la única posibilidad de implementar este esquema, no obstante, su uso es compatible con la librería utilizada en Python, phe [37]. Las fórmulas de Damgård-Jurik son similares, aplicando la parametrización, en vez de utilizar la precomputación de n^2, se utiliza n^{s+1}, donde s es el factor de expansión seleccionado.

Autenticación en Firebase

Para evitar escrituras no autorizadas en la base de datos, se ha creado un archivo de credenciales de Firebase para Python, y uno de propiedades con una pareja usuario/contraseña para Android. Estos archivos, si están presentes en el directorio correspondiente durante el lanzamiento del programa (en el caso de Android, la compilación del paquete), el sistema podrá enviar registros a la base de datos, de lo contrario, el sistema no enviará nada a la base de datos. Estos archivos de credenciales, por motivos de seguridad, no se incluyen en los repositorios, deben ir en el archivo *.gitignore*, de lo contrario, cualquier persona tendría acceso a ellos.

3.3.7. Problemas encontrados

Durante las pruebas manuales, se encontraron algunos problemas que se solucionaban de forma sencilla en algunos casos, mientras que otros presentaron más complejidad.

- Fallos en la API REST de Python. Estos errores venían relacionados de no comprobar si había un nodo activo, por lo que se creó un decorador para las peticiones de la API. Otros estaban relacionados con verificar si los parámetros introducidos eran correctos, se corrigieron con una validación de los campos introducidos.

- Problemas visuales en la aplicación. Con el tema oscuro, había problemas de contraste en la aplicación, se corrigió añadiendo variantes de los iconos y modificando los textos para que el sistema entendiera qué color debía mostrar.

- Pausa y reanudación del servicio en Android. El problema era que al reabrir la aplicación, la actividad principal no realizaba los cambios de textos pertinentes según si el servicio estuviera activo o no, resultando en una pantalla separada del servicio de red. Para corregirlo, se sobrescribió el método *onResume* que modifica los campos al volver a abrir la aplicación de la multitarea con la información que haya en el servicio en ese momento, o lo enciende si el sistema operativo lo ha mandado a dormir.

- Problemas con el HWM de ZeroMQ. En las pruebas de carga, los búffers de salida se llenaban hasta bloquear el sistema hasta que el sistema receptor consumiera los mensajes pendientes. Para evitarlo, cambié el envío de datos a un método no bloqueante, donde los mensajes que no quepa en el búffer, se pierden. Habitualmente, no se deberían perder mensajes por el límite de 2000 mensajes establecido para los búffers de entrada y salida, y la estrategia actual de encolar tareas entrantes en diferentes ejecutores, necesitando recursos mínimos para ello. Esto aumenta la presión en la memoria de emisores y receptores si se envían mensajes muy rápido. En los emisores, por almacenar mensajes en su búffer *ZMQ_SNDHWM*, que estarán en memoria hasta ser consumidos, y en los receptores, por la carga computacional de las operaciones y el tamaño de los mensajes recibidos.

4

Estudio experimental y resultados

Uno de los objetivos de este proyecto es comprobar la posibilidad de realizar intersecciones privadas entre dispositivos heterogéneos y medir su eficiencia computacional. Para tener la mayor variedad de métricas y datos, se han utilizado cuatro dispositivos, dos ordenadores que utilizan procesadores ARM, y dos dispositivos Android. Esto se ha decidido debido a que ARM representa el mercado de los bajos recursos, aunque las pruebas se podrían extender a arquitecturas x86.

Los dispositivos en particular son un Mac Studio (Apple M1 Max - 10 núcleos a 3.228 GHz - 32 GB), MacBook Pro (Apple M1 Pro - 8 núcleos a 3.228 GHz - 16 GB), un Samsung Galaxy S21 Ultra y Samsung Galaxy Tab S7 FE. Para las pruebas entre servicios web y dispositivos Android se usa el Mac Studio y el S21 Ultra.

Con el objetivo de simplificar los resultados para el lector, lo primordial es el procesador, en el caso de los dispositivos Mac, la velocidad será prácticamente igual por las limitaciones del Global Interpreter Lock (GIL) de Python[44], ya que ambos tienen la misma velocidad en sus núcleos de rendimiento [76], y macOS le concederá un núcleo de rendimiento al proceso de Python cuando note que es más pesado. En el caso de Android, presentan similares características y no se espera que la capacidad RAM influya en la asignación a la maquina virtual de Java.

4.1. Configuración del entorno

Se detalla la configuración de los sistemas previas a la ejecución de las pruebas. Destacar también que la red sobre las que se realizaron las pruebas es dedicada, no hay interrupciones de otros dispositivos a esta red durante la ejecución de las pruebas, aunque funciona en cualquier red de área local, como una red doméstica común.

Entorno Android: Sin multitarea y con actualizaciones y copias de seguridad desactivadas. La conexión a la red de área local se hace mediante el estándar Wi-Fi 6 (IEEE 802.11ax). La aplicación se lanza mediante el artefacto compilado, no desde Android Studio, aunque en algunos casos se utiliza ADB (*Android Debug Bridge*)[77] para comprobar que el sistema esté trazando los registros correctamente.

Entorno Servicios web: Se ha evitado utilizar el servidor de desarrollo de Flask para evitar tener una posible pérdida de eficiencia, ya que no es un servidor indicado para producción. Se ha utilizado un servidor ligero llamado Waitress [50], es un servidor WSGI [78] que solo utiliza Python y se puede usar en entornos de producción. Los sistemas tenían abierto una terminal para arrancar el servidor y Postman para las peticiones a la API REST. El sobremesa está conectado a la red de área local mediante ethernet, mientras que el portátil se conecta usando WiFi 6 (IEEE 802.11ax).

Entorno Global: Para realizar la conexión inicial, se ha añadido al compañero en concreto, utilizando los identificadores de cada dispositivo y se ha comprobado que se reconocen. Después se han lanzado las pruebas con diferentes configuraciones. Estas operativas se pueden ver en las Figuras 4.1 y 4.2 para los servicios web y los dispositivos Android, respectivamente.

4.1.1. Configuración de Experimentos

Para realizar los experimentos, se han combinado los diferentes sistemas, implementaciones y protocolos, para tener una visión global del consumo de tiempo y recursos de cada dispositivo. Las pruebas se han lanzado indicando en la petición de la API los parámetros a utilizar, y en Android desde la opción de lanzar tests en la hoja de opciones del dispositivo seleccionado. Presentes en la imagen inferior de las Figuras 4.1 y 4.2, respectivamente. Se han hecho de forma que se tenga una muestra considerable de cada paso de las intersecciones utilizando los distintos protocolos y criptosistemas integrados.
Configuración de los parámetros: Dominio: 500, Conjunto de datos: 50, Iteraciones: 1000, Tamaño de clave: 2048, Factor de expansión (Damgård-Jurik) s: 2.

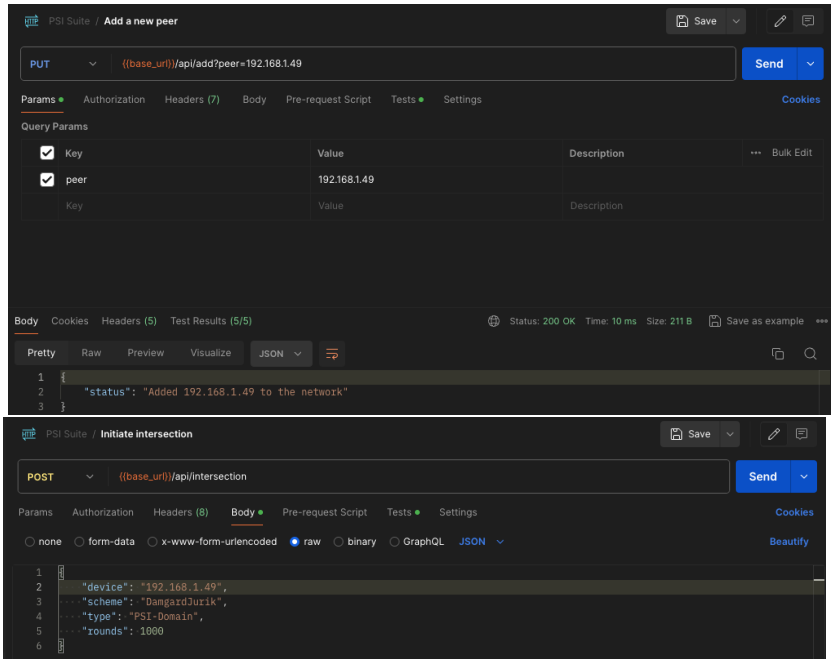

Figura 4.1: Añadir compañero mediante la API (arriba). Lanzamiento de tests (abajo)

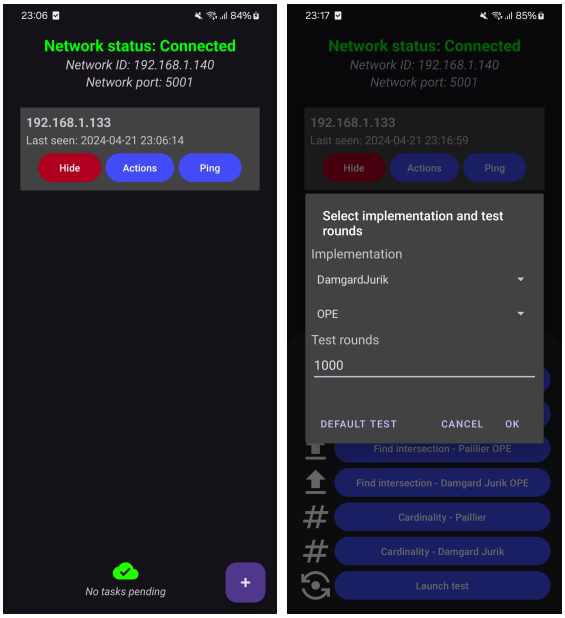

Figura 4.2: Vista de un compañero de la red (Columna Izquierda). Lanzamiento de tests (Columna derecha)

Por motivos de tiempo no se ha probado con claves de 4096 bits, ya que 2048 se sigue considerando seguro para criptosistemas basados en factorización de enteros según las recomendaciones del NIST [79], en 2024. No obstante, se han realizado pruebas con claves de 1024 y 512, que se pueden encontrar en el Apéndice A.5.

Para reducir al mínimo el tiempo de la captura de datos para el registro, el

tiempo se captura antes de que se procesen, asegurando que únicamente se mide el tiempo que ha tardado en ejecutarse la función. La Firebase Realtime Database se limpia cada vez que se realiza un test, descargando previamente sus contenidos en JSON, de forma que se tienen todas las muestras segregadas dependiendo por el tipo de dispositivo, criptosistema y protocolo utilizado.

La concurrencia es importante en este estudio, dependiendo de la velocidad en la que el emisor realice sus operaciones iniciales, el receptor tendrá que estar manejando una ingesta de mensajes mayor, afectando a recursos de otros hilos, pudiendo resultar en distintos tiempos de operación. El factor conectividad también es relevante, si por cualquier razón ajena al sistema, este deja de recibir mensajes, no se pierden (si no llega al límite *SNDHWM* del emisor), se almacenarán en el búffer de salida del emisor, si el sistema receptor recupera la conexión, consumirá todos los mensajes acumulados en búffer del emisor, esto puede reflejarse en el emisor como mayor consumo de memoria, y en el tiempo necesario para terminar la prueba.

4.1.2. Resultados

A continuación, se presentan los resultados de las diferentes pruebas realizadas, así como su interpretación. Para poder analizar los resultados de los archivos JSON generados por la base de datos no relacional de Firebase, se ha creado un script de Python utilizando Pandas [1] para el procesado de los datos de los archivos JSON y Matplotlib [2] para crear gráficos basados en los datos procesados.

Las tablas 4.1 y 4.2 muestran los datos más relevantes relacionados con el tiempo que se tarda en ejecutar cada paso de las intersecciones, dependiendo de si el dispositivo que inicia el proceso es un servicio web o un dispositivo Android. Los tiempos para las operaciones son similares para cada dispositivo independientemente de con quién interactúe, por lo que los resultados presentados se centran en dispositivos heterogéneos y dan medidas aplicables tanto a servicios web como a dispositivos Android.

Los resultados se dividen en tres pasos: cifrado, evaluación y descifrado, como se ve en las tablas a continuación. El cifrado y descifrado se ejecutan en la parte interesada en el cálculo de la intersección, es decir, quien hace la operación de cifrado es quien envía sus datos a evaluar y el receptor de estos datos evaluará el contenido cifrado contra su contenido utilizando una de las técnicas propuestas. La lógica de los tres pasos varía según el protocolo utilizado. WS significa *Web Service* o Servicio Web y se utiliza para representar que es el sistema de Python el que realiza la acción.

[1] https://pandas.pydata.org/
[2] https://matplotlib.org/

Tiempos de cómputo

Criptosistema	Protocolo	Cifrado (WS)	Evaluación (Android)	Descifrado (WS)	Total
Paillier	Según dominio	36,7252	6,8319	6,7785	50,3356
	OPE	2,8232	5,5420	0,9754	9,3406
	OPE - c	3,0201	7,0190	0,6959	10,7350
Damgård-Jurik	Según dominio	168,0163	24,7288	131,6832	324,4283
	OPE	13,3630	12,9425	13,7508	40,0562
	OPE - c	14,6087	9,1552	13,4666	37,2304

Tabla 4.1: Servicio web inicia el proceso de intersección contra un dispositivo Android. Resultados - Claves de 2048 bits - Tiempo en segundos

Criptosistema	Protocolo	Cifrado (Android)	Evaluación (WS)	Descifrado (Android)	Total
Paillier	Según dominio	18,3776	19,5979	36,5151	74,4906
	OPE	2,2997	7,5709	2,6425	12,5131
	OPE - c	2,0926	7,2716	2,4494	11,8136
Damgård-Jurik	Según dominio	77,1107	64,8301	78,8486	220,7894
	OPE	8,0248	5,3870	7,3135	20,7253
	OPE - c	8,5634	4,0684	7,8600	20,4918

Tabla 4.2: Dispositivo Android inicia el proceso de intersección contra un servicio web. Resultados - Claves de 2048 bits - Tiempo en segundos

El protocolo OPE y su variante de cardinalidad OPE-c son considerablemente más rápidos, por lo menos con esta configuración, mientras que según el dominio son más lentas, esto se debe a que cifran un valor por cada elemento del dominio. También se ve cómo Damgård-Jurik es considerablemente más lento que Paillier por el espacio de cifrado configurado, estos resultados nos dan a entender que tendríamos que sacrificar este factor de expansión si queremos más rendimiento, cosa que afecta a la seguridad al reducir el espacio de prueba en un ataque a fuerza bruta.

En una aplicación real con estos criptosistemas, requiere encontrar un equilibrio mayor entre seguridad y eficiencia si se buscase rendimiento constante y encontrar algoritmos más eficientes. Por otro lado, pongamos una granja IoT en la que se quiere que los sensores realicen intersecciones entre los datos que hayan ido recopilando, o con un servidor, esto no se haría constantemente sino cuando la recopilación fuera alta, y el intercambio tardaría como se puede comprobar en la Tabla 4.1, pero todo el tráfico de la red quedaría asegurado.

Como análisis exploratorio, también se han realizado pruebas con una librería en Python con la implementación de Brakerski-Fan-Vercauteren [41]. Se ha evaluado un protocolo en que funciona este criptosistema usando intersecciones calculadas según el dominio. Se ha realizado una prueba de 1000 iteraciones sobre los dos dispositivos Mac que ha otorgado buenos resultados como se ve en la Figura 4.3.

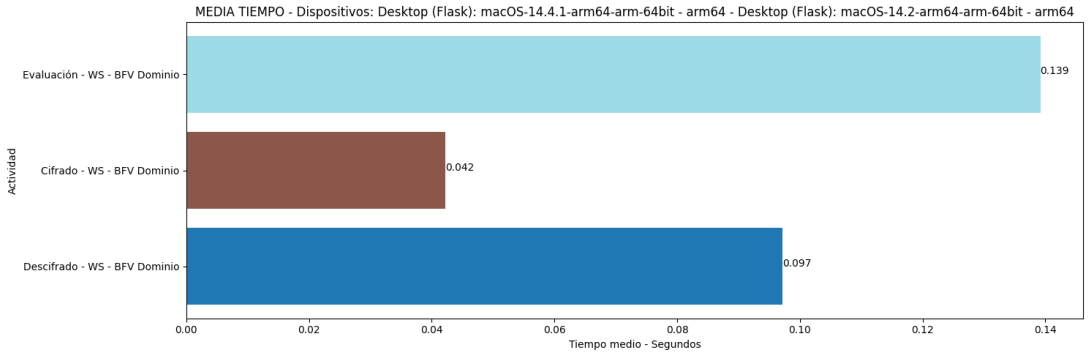

Figura 4.3: Media de tiempo por operación BFV - Servicio Web

Estos resultados indican mucha rapidez, pero hay que tener en cuenta que al presentarlo como una prueba de concepto, no se ha mirado por la seguridad, y no es directamente comparable con la longitud de claves convencional de los criptosistemas basados en factorización de enteros.

Se han hecho pruebas con parámetros simples $q = 288230376134934529$, $n = 2$ y $t = 2017$, pero son difíciles de configurar, ya que valores como la t dependen del método de codificación que utilice la implementación [80], estos valores funcionan con las intersecciones según el dominio, pero en esquemas como OPE frecuentemente introducen demasiado ruido, luego como mejora a futuro habría que intentar extenderlo a las recomendaciones de la cita, ya que el aumento actual de ruido tras una operación hace que no se puedan descifrar los valores originales o esperados [31].
Es esperable que utilizar valores mayores aumente la seguridad, y la velocidad debería seguir siendo rápida. La propuesta de algoritmo de intercambio de claves Crystals-Kyber [81] fue seleccionada como ganador para el proyecto de criptografía post-cuántica del NIST en el intercambio de claves [82]. Es un algoritmo que también se basa en el problema del aprendizaje con errores (LWE) y destaca por su rapidez [83].

Generación de claves

Se ha realizado esta prueba para contrastar la velocidad de las librerías que se han utilizado en Python con las de Android, especialmente en el caso de Damgård-Jurik, al utilizar primos seguros (un primo p tal que $(p-1)/2$ también es primo, y la realización de cálculos para poder compartir la clave privada, cosas que no realiza la implementación de Android, y que se quiere comprobar como afecta al aumentar la longitud de la clave. También se incluyen los resultados de Android. La generación de claves es similar en ambos casos.

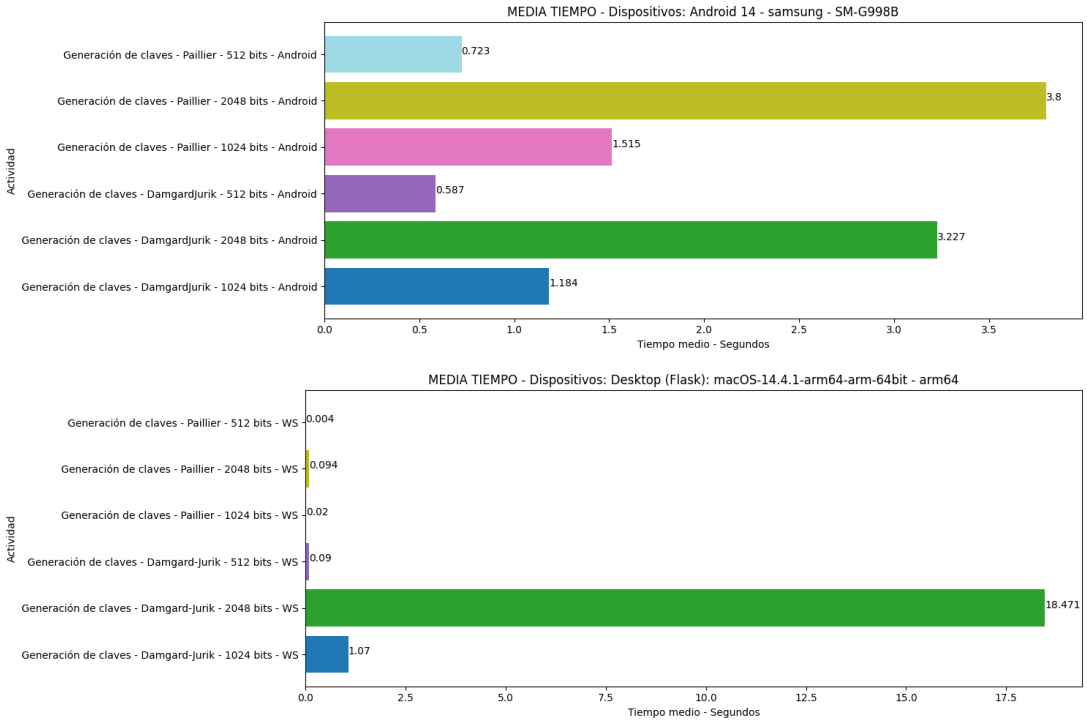

Figura 4.4: Tiempos de generación de claves - Android (arriba), Servicio Web (abajo)

En la Figura 4.4 se puede ver cómo el uso de primos seguros hace que el promedio de generación de claves aumente considerablemente cuando se generan claves de 2048 bits. En Android, los valores se mantienen estables, con la diferencia de que Damgård-Jurik tarda un poco más, debido al factor de expansión seleccionado $s = 2$. Los resultados de Android son más lentos que Python, probablemente por el uso de números aleatorios seguros de Java.

Tamaño de textos cifrados

Esta comparación se puede hacer únicamente entre dispositivos del mismo tipo, ya que cada lenguaje maneja la memoria de forma distinta, no obstante, sirve para comparar los recursos de memoria que se utilizan con cada criptosistema en cada dispositivo. En Android, se usa "Parcel", que da una estimación del tamaño de la representación serializada del objeto en bytes que está consumiendo dicha *Parcel* [84], pero no tiene relación directa con su peso físico real, para más precisión habría que hacer una multiplicación por el valor de los elementos, cosa más compleja, así como determinar el tamaño de los objetos en la lista. En Python, se usa *sys.getsizeof()* da el tamaño en memoria del objeto que se le pasa (la lista de cifrados) [85].

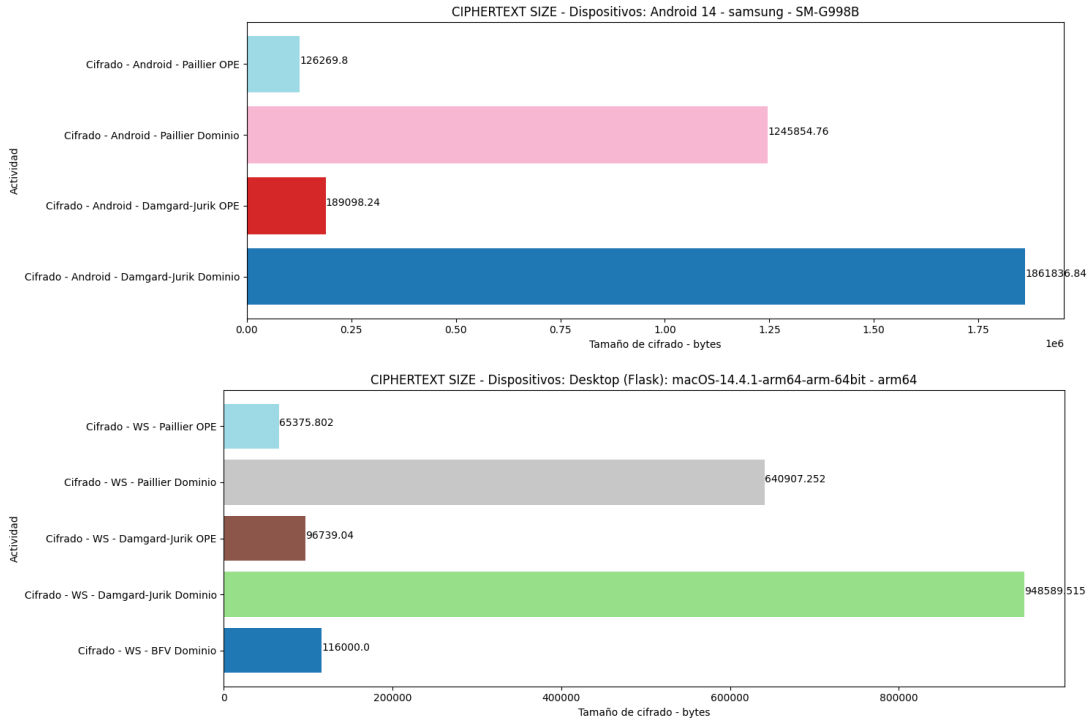

Figura 4.5: Tamaño de textos cifrados en memoria - Android (arriba) - WS (debajo) - Longitud de claves: 2048 bits

En la figura 4.5 se puede comprobar como las intersecciones basadas en un dominio son más pesadas ya que su naturaleza cifra un mensaje por cada valor del dominio. De nuevo, el factor de expansión de Damgård-Jurik influye, generando mensajes más grandes.

Recuperación ante errores

La forma de envío de los registros permite evaluar la ejecución pruebas e identificar posibles recuperaciones automáticas"tras fallos. Algunas pruebas se realizaron durante la noche, revisándose los resultados posteriormente. En una prueba, se observó un fallo de conectividad en uno de los dispositivos, el cual, al reconectarse, consumió los mensajes acumulados en el búffer del emisor.

La Figura 4.6 muestra la prueba de Damgård-Jurik entre un servicio web (M1 Max) y un dispositivo Android (S21 Ultra). El eje X representa la marca en el tiempo de ejecución en que se realizó cada operación (que está representada por un punto), el eje Y representa el tiempo que le llevó al sistema terminar dicha operación.

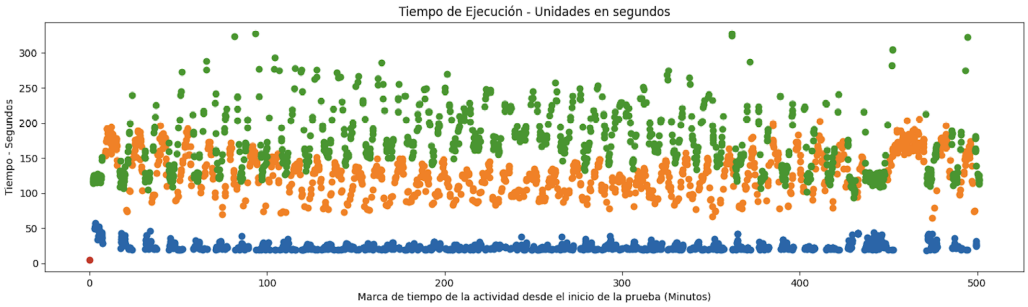

Figura 4.6: Damgård-Jurik-2048 - Gráfico de ejecución de actividades y tiempo necesario entre servicio web y Android - Generación de clave (rojo), cifrado (verde), evaluación (azul) y descifrado (naranja)

En este gráfico, se evidencia una pérdida de conectividad alrededor del minuto 430. El dispositivo Android dejó de consumir mensajes, mientras el servicio web continuó enviando mensajes. Al reconectarse el dispositivo Android, procesó los mensajes pendientes, y el servicio web los manejó como debía al recibirlos. En el final del gráfico, las últimas operaciones son del dispositivo Android evaluando el mensaje recibido, esto se debe a que el ordenador estaba programado para apagarse en ese momento, y no llegó a terminar la prueba, quedándose en 972 registros de los 1000 iniciales. Se comprueba el funcionamiento de la prioridad en las operaciones concurrentes. Los puntos naranjas y verdes del servicio web indican que el sistema prioriza procesar los mensajes recibidos antes que las tareas pendientes para liberar los mensajes recibidos de memoria antes, ya que ocupan más, realizando los cálculos finales junto a los iniciales cuando hay mensajes de este tipo. En dispositivos IoT sería algo crucial para no quedarnos sin memoria.

Se puede ver también que cuando una operación parece tardar más, está correlacionada con otra que ha tardado menos. Probablemente debido a que el Global Interpreter Lock (GIL) de Python ha concedido más tiempo a una operación o ha evitado un cambio de contexto para mejorar el rendimiento.

Consumo de memoria

En los servicios web, Python reutiliza la memoria que ha sido liberada por el recolector de basura pero dependiendo de cada sistema, puede preferir asignar nueva memoria. La figura 4.7 representa una prueba realizada entre el sistema de 32GB de RAM, frente al de 16GB. En el primero el sistema operativo y el intérprete es probable que estén siendo más laxos con la liberación de memoria, resultando en que la memoria se mantenga asignada más tiempo, aunque no se esté usando, frente a las métricas reportadas por el sistema de 16GB, donde se ve cómo libera memoria periódicamente. La generación de clave es nula porque su tiempo de ejecución hizo que no le diera tiempo al sistema a medir la RAM.

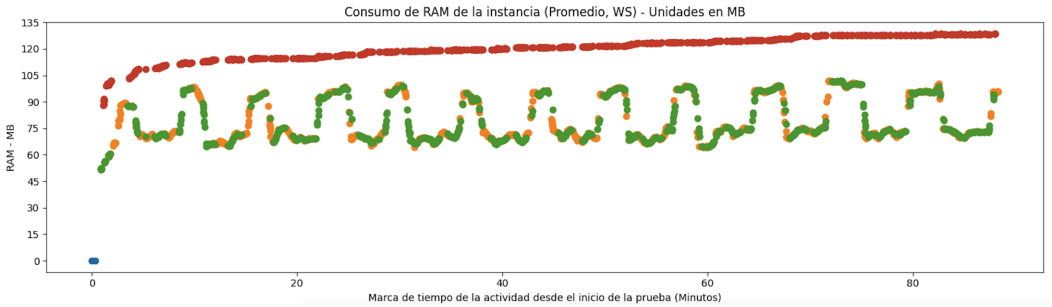

Figura 4.7: Uso de RAM en la ejecución de una prueba - Mac - Generación de clave (azul), cifrado (verde), evaluación (rojo) y descifrado (naranja)

En Android, dependemos de lo que el Tiempo de Ejecución de Android (ART) [74] pueda conceder a la máquina virtual de la aplicación para la asignación física de la memoria. La Figura 4.8 muestra el uso de RAM en Android cuando los mensajes son grandes y difíciles de procesar, como en el caso de las intersecciones según el dominio utilizando Damgård-Jurik, y cuando son más ligeros, como utilizando OPE-c con Paillier.

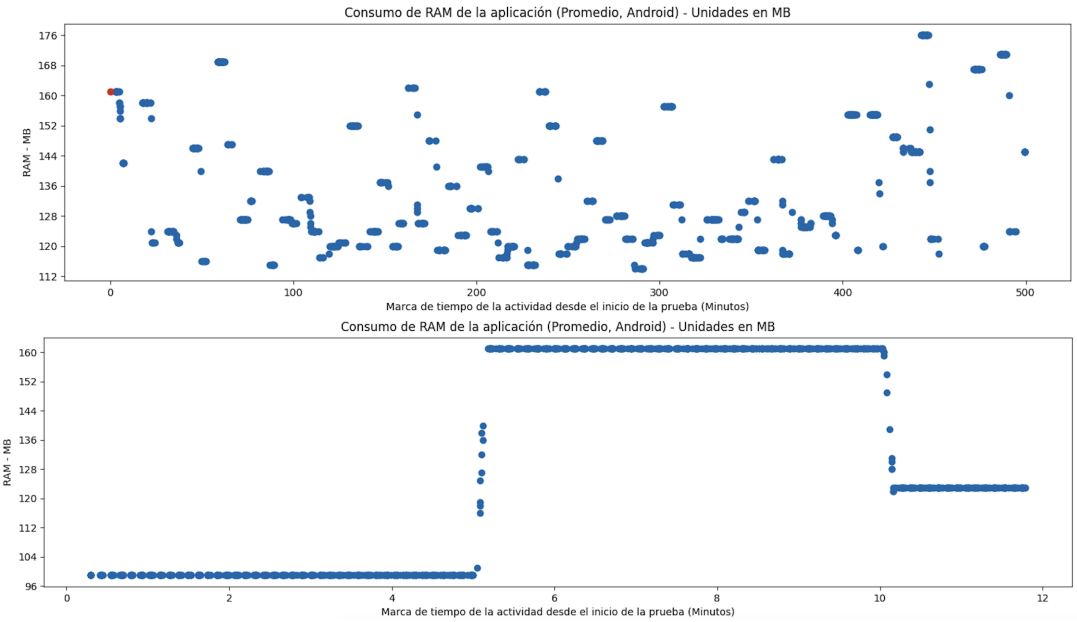

Figura 4.8: Uso de RAM en la ejecución de una prueba - Evaluación PSI según dominio (puntos azules) - Damgård-Jurik (arriba) - Evaluación OPE-c (puntos azules) - Paillier (abajo) - Android

Este gráfico muestra cómo el ART ajusta la asignación de memoria según las necesidades de la aplicación y el sistema en sí. Al recibir mensajes más grandes que requieren una ejecución más exigente, el ART realiza ciclos de recolección de basura más constantes, resultando en la oscilación que se ve en el uso de memoria en el gráfico superior.

Las ventajas de este enfoque son la eficiencia y adaptabilidad del manejo de memoria por parte de Android. El sistema asigna memoria dinámicamente dependiendo de lo que la aplicación necesite.

La principal desventaja es lo impredecible que es la asignación de memoria en estas situaciones, dificultando conocer con certeza el comportamiento del sistema en situaciones de alta demanda. Esta imprevisibilidad puede llevar a variaciones en el rendimiento de la aplicación y el sistema en general, el nodo al final corre un servicio Android, lo que puede hacer que el dispositivo en general proporcione un peor rendimiento mientras este esté corriendo y bajo estrés.

Con mensajes más pequeños y que requieren menos recursos para completarse, el ART le concederá a la máquina virtual de la aplicación una memoria y la recolección de basura será más infrecuente (de hecho, esto es una de las ventajas del ART), ya que el sistema puede no necesitarla en ese momento, y sería más rápido para la aplicación poder acceder a su asignación directamente si la necesidad de solicitarla al ART.

Otras métricas

El tiempo de CPU en Android se ha comprobado que varía en cada operación debido a la ejecución de tareas por varios hilos. El cambio de contexto entre hilos introduce una sobrecarga adicional, incluyendo el tiempo necesario para guardar y cargar el estado del hilo, así como la invalidación de caché frecuente. La espera para escribir en el búffer de salida ZMQ, que requiere una escritura *thread-safe*, evitando una condición de carrera, también afecta a este tiempo.

En Python, el intérprete está siempre ocupado debido al Global Interpreter Lock (GIL). Las lecturas del uso de CPU de la instancia muestran un 100 % , indicando que el interpretador está siempre ocupado. El uso de la CPU al completo es menor, alrededor del 18 %, ya que el GIL limita el rendimiento multihilo.

5

Conclusiones y trabajos futuros

Al haber terminado el proyecto considero que se han logrado los principales objetivos que se buscaban:

Por una parte, he investigado tanto la criptografía homomórfica como las distintas opciones para calcular intersecciones de conjuntos privados. La investigación se complementa con la implementación de dos esquemas criptográficos y un estudio experimental para comparar su rendimiento. Por otra parte, se ha desarrollado un sistema multiplataforma para la evaluación de diferentes esquemas criptográficos que permite que dispositivos heterogéneos intercambien mensajes cifrados en una red descentralizada y ofrezcan una muestra realista de recursos disponibles para destinar a estas operaciones, como un dispositivo móvil.

El trabajo ha supuesto una sucesión de retos a medida que avanzaba. El primero fue comprender cómo aplicar la criptografía homomórfica al cálculo de PSI, la búsqueda y compresión de bibliografía en el campo, que añade complejidad al trabajo. Esto me permitió comprender conceptos avanzados de criptografía especialmente en los esquemas investigados Paillier, Damgård-Jurik y Brakerski Fan-Vercauteren, y cómo se pueden aprovechar sus propiedades criptográficas para proteger la privacidad de los datos. Los dos primeros esquemas cuentan con implementaciones propias, que demuestran buenas propiedades para PSI, como se puede comprobar mediante los mecanismos de registro implementados en la plataforma que han permitido comparar el rendimiento y consumo de recursos.

El desarrollo multiplataforma de la aplicación Android y los servicios web en Python me han ayudado a profundizar en el desarrollo de servicios web con APIs REST y en el desarrollo Android. El desarrollo de Android permite realizar las

mismas operaciones pero en un dispositivo con recursos más limitados, para comprobar qué tipo de implicaciones tendría la implementación de mecanismos que asegurasen la privacidad en estos dispositivos. La coordinación entre el desarrollo de servicios web y de Android ha sido muy importante para que ambos sistemas ofrecieran las mismas funcionalidades, de acuerdo con los requisitos expuestos en este documento.

Durante el desarrollo del proyecto considero que he aplicado todos los conocimientos impartidos en el grado, desde asignaturas orientadas a la gestión de proyectos, hasta las asignaturas que enseñan patrones y buenas prácticas, pasando por asignaturas matemáticas, lógicas y, por la naturaleza del sistema, redes. Todo ello se emplea aquí en un intento de realizar un trabajo profesional y documentado de forma acorde a un proyecto de software. Este proyecto ha alcanzado los objetivos establecidos, y ha proporcionado una plataforma sólida para futuras investigaciones y desarrollos en el campo de las intersecciones privadas de conjuntos, contribuyendo al avance del conocimiento en un área más necesaria que nunca.

5.1. Trabajos futuros

Por cuestiones del tiempo limitado del que se dispone para realizar el proyecto, el alcance se limita, lo que da paso a algunos trabajos futuros, que se han valorado, pero no se han introducido en el proyecto.

Como se ha presentado en la sección de pruebas, *la migración a enfoque multiproceso en Python* se ha valorado hacer, pero se descartó teniendo en cuenta que el objetivo era que dispositivos de bajos recursos pudieran calcular conjuntos de intersecciones privadas de manera eficiente, tampoco afectaba del todo, puesto que ningún dispositivo de capacidades reducidas iba a tener un sistema como en los que se ha probado. En cualquier caso, es una mejora que permitiría evitar las limitaciones del GIL de Python, y en algunos casos mejorar el rendimiento.

Se valoró *añadir más criptosistemas y protocolos PSI* pero la complejidad técnica de ambos desarrollos hizo que se limitara este apartado. Esto permitiría comparar las mediciones que ya se tienen con otras nuevas, para decidir, por ejemplo, cuál podría ser la mejor elección para algún uso específico en el que primen los bajos recursos. Con la investigación que hay actualmente, sería interesante implementar criptosistemas candidatos para la resistencia post-cuántica, NTRU [86] con su cifrado basado en retículas [87], o Cheon-Kim-Kim-Song (CKKS) [32] con su cifrado, como BFV, basado en el anillo de aprendizaje con errores [40] y el aprendizaje con errores [39]. Se podría terminar la propuesta de implementación del criptosistema Brakerski-Fan-Vercauteren (BFV) para Python y diseñarla en Android, hasta el momento no se ha podido romper con algoritmos post-cuánticos conocidos.

En cuanto a protocolos PSI la carga de trabajo de comprobar la compatibilidad con las operaciones homomórficas que pueden realizar los criptosistemas, hizo que se quedasen en los tres protocolos presentados. Algún protocolos extra a estudiar podría ser el protocolo básico de *Oblivious Transfer (OT)* [88].

En cuanto a la plataforma, *diseñar un sistema «backend» para los logs* dedicado a guardar las métricas en una base de datos dedicada habría sido una mejor opción para poder realizar consultas SQL. La función de la RealtimeDatabase de Firebase ha cumplido con lo que se quería en términos de registro pero ha requerido desarrollar un analizador para archivos JSON. Al no tener conectores con bases de datos en Android, se requiere de un «backend» que proporcione un servicio web al que poder hacer las peticiones. La arquitectura permite un cambio sencillo, y quedaría por determinar la autenticación que se fuera a realizar. El sistema está abierto a mejoras futuras, también se podría implementar una base de datos local.

Como se comentó anteriormente, el desarrollo incluye los archivos necesarios para generar una imagen de Docker de los servicios web. De hecho, se puede comprobar el funcionamiento con *docker compose up*, que levantará cuatro servidores de Waitress con su dirección y cuatro nodos en la misma red de Docker. El problema que tiene es que no pueden registrar un uso de CPU. Por falta de tiempo, no se diagnosticó la causa, pudiendo ser simplemente de la virtualización por el sistema en el que se ha probado. No obstante, se propone como mejora, porque en Raspberrys u otros dispositivos de bajos recursos, sería interesante ver cómo se comportan, y para ello es mucho más fácil desplegar un contenedor de Docker con la imagen.

Finalmente, en Android se implementaron algunas notificaciones en el «NetworkService», pero se quedó ahí el desarrollo por falta de tiempo, el objetivo era que todo los errores reportasen mediante notificación, de manera que el «NetworkService» era el punto del sistema que se encargaría de todas las notificaciones, y no solo de las operaciones que reportase el «LogService», como se hace actualmente. De manera similar, se planteó sin materializarse, el poder tener eventos en el frontal del servicio web, de forma que hubiera notificaciones como las de Android o reporte de errores. Fue algo que se evaluó, pero no se llegó a tiempo para la implementación total que se buscaba, por lo que se mantuvo únicamente el notificado de actividades en Android.

Bibliografía

[1] S. D. Warren and L. D. Brandeis, "The right to privacy," *Harvard Law Review*, vol. 4, no. 5, pp. 193–220, 1890. [Online]. Available: http://www.jstor.org/stable/1321160

[2] F. O'Sullivan, "What is a passkey?" Apr 2024. [Online]. Available: https://proton.me/blog/what-is-a-passkey

[3] [Online]. Available: https://www.cloudflare.com/en-gb/learning/ssl/what-is-an-ssl-certificate/

[4] D. Morales, I. Agudo, and J. Lopez, "Private set intersection: A systematic literature review," *Computer Science Review*, vol. 49, p. 100567, 2023.

[5] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–19.

[6] A. Heinrich, M. Hollick, T. Schneider, M. Stute, and C. Weinert, "{PrivateDrop}: Practical {Privacy-Preserving} authentication for apple {AirDrop}," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3577–3594.

[7] . . p. U. Dan Goodin Jan 12, "Apple airdrop leaks user data like a sieve. chinese authorities say they're scooping it up." Jan 2024. [Online]. Available: https://arstechnica.com/security/2024/01/hackers-can-id-unique-apple-airdrop-users-chinese-authorities-claim-to-do-just-that/

[8] G. Sánchez-Arias, C. González García, and B. Pelayo García-Bustelo, "Midgar: Study of communications security among smart objects using a platform of heterogeneous devices for the internet of things," *Future Generation Computer Systems*, vol. 74, 02 2017.

[9] [Online]. Available: https://www.sciencedirect.com/topics/computer-science/paillier-cryptosystem

[10] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography*, K. Kim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 119–136.

[11] C. Y. Kim, B. C. Ng, J. Sahni, N. Samian, and W. K. Seah, "Blockchain network platform for iot data integrity and scalability," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, 2023, pp. 162–171.

[12] M. Mukhandi, "Decentralised and scalable security for iot devices," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 411–412. [Online]. Available: https://doi.org/10.1145/3485730.3492901

[13] S. Cherrier, Z. Movahedi, and Y. M. Ghamri-Doudane, "Multi-tenancy in decentralised iot," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 256–261.

[14] F. Corno and L. Mannella, "Security evaluation of arduino projects developed by hobbyist iot programmers," *Sensors*, vol. 23, no. 5, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/5/2740

[15] L. Kissner and D. Song, "Privacy-preserving set operations," in *Advances in Cryptology – CRYPTO 2005*, V. Shoup, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 241–257.

[16] W. Clark, "Private set intersection with the paillier cryptosystem," Oct 2020. [Online]. Available: https://blog.openmined.org/private-set-intersection-with-the-paillier-cryptosystem/

[17] P. D'ARCO, M. I. Gonzalez Vasco, A. L. PÉEZ DEL POZO, C. Soriente, and R. Steinwandt, "Private set intersection: New generic constructions and feasibility results." *Advances in mathematics of communications*, vol. 11, no. 3, 2017.

[18] "apple.com," https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf, 2021.

[19] A. Bhowmick, D. Boneh, S. Myers, K. Talwar, and K. Tarbe, "apple.com," https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf, 2021.

[20] T. Hardwick, "Apple provides further clarity on why it abandoned plan to detect csam in icloud photos," Sep 2023. [Online]. Available: https://www.macrumors.com/2023/09/01/apple-explains-csam-plan-abandoned/

[21] A. Support, "Password Monitoring — support.apple.com," https://support.apple.com/en-gb/guide/security/sec78e79fc3b/web.

[22] H. Chen, H. Chen, Z. Zhang, H. Xiong, and A. Raghunathan, "How meta is improving password security and preserving privacy," Aug 2023. [Online]. Available: https://engineering.fb.com/2023/08/08/security/how-meta-is-improving-password-security-and-preserving-privacy/

[23] "Semi-honest Adversary — wiki.mpcalliance.org," https://wiki.mpcalliance.org/semi_honest_adversary.html.

[24] "Private set intersection with the paillier cryptosystem"," https://blog.openmined.org/private-set-intersection-with-the-paillier-cryptosystem/, 2020.

[25] [Online]. Available: https://cseweb.ucsd.edu/~mihir/cse209B-Wi23/arte.pdf

[26] D. Morales, I. Agudo, and J. Lopez, "Private set intersection: A systematic literature review," *Computer Science Review*, vol. 49, p. 100567, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013723000345

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[28] "Cryptography academy." [Online]. Available: https://cryptographyacademy.com/elgamal/

[29] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Advances in Cryptology — EUROCRYPT '97*, W. Fumy, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 103–118.

[30] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," Cryptology ePrint Archive, Paper 2013/075, 2013, https://eprint.iacr.org/2013/075. [Online]. Available: https://eprint.iacr.org/2013/075

[31] Feb 2022. [Online]. Available: https://inferati.com/blog/fhe-schemes-bgv

[32] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," Cryptology ePrint Archive, Paper 2016/421, 2016, https://eprint.iacr.org/2016/421. [Online]. Available: https://eprint.iacr.org/2016/421

[33] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[34] G. Bonnoron, C. Fontaine, G. Gogniat, V. Herbert, V. Lapôtre, V. Migliore, and A. Roux-Langlois, "Somewhat/fully homomorphic encryption: Implementation progresses and challenges," in *Codes, Cryptology and Information Security*, S. El Hajji, A. Nitaj, and E. M. Souidi, Eds. Cham: Springer International Publishing, 2017, pp. 68–82.

[35] W. Clark, "What is the paillier cryptosystem?" Oct 2020. [Online]. Available: https://blog.openmined.org/the-paillier-cryptosystem/

[36] S. Kim and J. Jin, "Intel paillier cryptosystem library." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/homomorphic-encryption/iso-compliant-paillier-cryptosystem-library.html

[37] "GitHub - data61/python-paillier: A library for Partially Homomorphic Encryption in Python — github.com," https://github.com/data61/python-paillier.

[38] "GitHub - cryptovoting/damgard-jurik: A Python implementation of the threshold variant of the Damgard-Jurik cryptosystem. — github.com," https://github.com/cryptovoting/damgard-jurik.

[39] [Online]. Available: https://hmn.wiki/es/Learning_with_errors

[40] [Online]. Available: https://hmn.wiki/es/Ring_learning_with_errors

[41] Sarojaerabelli, "Sarojaerabelli/py-fhe: A python library for fully homomorphic encryption." [Online]. Available: https://github.com/sarojaerabelli/py-fhe/tree/master

[42] I. Sommerville, *Ingeniería del software*. Pearson educación, 2005.

[43] https://flask.palletsprojects.com/.

[44] "Globalinterpreterlock python wiki." [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock

[45] "ZeroMQ — zeromq.org," https://zeromq.org/.

[46] "Home — docker.com," https://www.docker.com/.

[47] "oogle issue tracker." [Online]. Available: https://issuetracker.google.com/issues/37140047

[48] "unittest; Unit testing framework Python 3.9.18 documentation," https://docs.python.org/3.9/library/unittest.html.

[49] "JUnit &x2013; About — junit.org," https://junit.org/junit4/.

[50] "GitHub - Pylons/waitress: Waitress - A WSGI server for Python 3 — github.com," https://github.com/Pylons/waitress/.

[51] "Firebase — Google's Mobile and Web App Development Platform — firebase.google.com," https://firebase.google.com/.

[52] "Firebase Realtime Database — firebase.google.com," https://firebase.google.com/docs/database/.

[53] "Crashlytics App Crash & Stability Reporting — Firebase — firebase.google.com," https://firebase.google.com/products/crashlytics.

[54] "Mockito framework site — site.mockito.org," https://site.mockito.org/.

[55] "psutil — pypi.org," https://pypi.org/project/psutil/.

[56] "Download Android Studio & App Tools - Android Developers — developer.android.com," https://developer.android.com/studio.

[57] "PyCharm: the Python IDE for data science and web development — jetbrains.com," https://www.jetbrains.com/pycharm/.

[58] "Postman API Platform — postman.com," https://www.postman.com/.

[59] "Git — git-scm.com," https://git-scm.com/.

[60] "GitHub Desktop — desktop.github.com," https://desktop.github.com/.

[61] "StarUML — staruml.io," https://staruml.io/.

[62] "Flowchart Maker & Online Diagram Software — app.diagrams.net," https://app.diagrams.net/.

[63] [Online]. Available: https://grpc.io/

[64] DazWilkin, "Which option is more suitable for microservice? GRPC or Message Brokers like RabbitMQ — stackoverflow.com," https://stackoverflow.com/a/69420063, 2021.

[65] "RabbitMQ: One broker to queue them all — RabbitMQ — rabbitmq.com," https://www.rabbitmq.com/.

[66] S. Kurade, "Comparison of tcp, (rmq), mqtt, redis, and grpc transport layers in nest.js microservices," Feb 2023. [Online]. Available: https://blog.nonstopio.com/comparison-of-tcp-rmq-mqtt-redis-and-grpc-transport-layers-in-nest-js-microservices-2c10a2a98a6f

[67] "MQTT - The Standard for IoT Messaging — mqtt.org," https://mqtt.org/.

[68] "Get started — zeromq.org," https://zeromq.org/get-started/.

[69] K. Wisniowski, "Rabbitmq vs zeromq what's the difference? (pros and cons)," Aug 2022. [Online]. Available: https://cloudinfrastructureservices.co.uk/rabbitmq-vs-zeromq-whats-the-difference/

[70] D. Walsh, D. founded Smart Home Perfected in 2019 as a personal passion project. With a Bachelor of Science (Hons) Degree in Computing, and I. from The Open University, "The definitive zigbee guide," Apr 2024. [Online]. Available: https://www.smarthomeperfected.com/zigbee/

[71] P. Hintjens, *ZeroMQ: Messaging for Many Applications*, ser. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2013. [Online]. Available: https://books.google.es/books?id=KWtT5CJc6FYC

[72] [Online]. Available: https://stackshare.io/stackups/mqtt-vs-zeromq

[73] [Online]. Available: https://www.bittorrent.org/introduction.html

[74] "Android runtime and Dalvik — Android Open Source Project — source.android.com," https://source.android.com/docs/core/runtime.

[75] "Android developers." [Online]. Available: https://developer.android.com/guide/topics/manifest/application-element

[76] "Compared: M1 vs M1 Pro and M1 Max — AppleInsider — appleinsider.com," https://appleinsider.com/articles/21/10/30/compared-m1-vs-m1-pro-and-m1-max.

[77] "Android Debug Bridge (adb) — Android Studio — Android Developers — developer.android.com," https://developer.android.com/tools/adb.

[78] "PEP 3333 – Python Web Server Gateway Interface v1.0.1 — peps.python.org — peps.python.org," https://peps.python.org/pep-3333/.

[79] D. Giry, "Keylength - NIST Report on Cryptographic Key Length and Cryptoperiod (2020) — keylength.com," https://www.keylength.com/en/4/.

[80] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.

[81] "Kyber — pq-crystals.org," https://pq-crystals.org/kyber/index.shtml.

[82] I. T. L. Computer Security Division, "Selected algorithms 2022 - post-quantum cryptography: Csrc." [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022#

[83] Jul 2022. [Online]. Available: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms

[84] "Parcel — Android Developers — developer.android.com," https://developer.android.com/reference/android/os/Parcel#dataSize().

[85] "sys &x2014; System-specific parameters and functions &x2014; Python 3.9.19 documentation — docs.python.org," https://docs.python.org/3.9/library/sys.html#sys.getsizeof.

[86] [Online]. Available: https://ntru.org/

[87] D. P. Chi, J. W. Choi, J. S. Kim, and T. Kim, "Lattice based cryptography for beginners," Cryptology ePrint Archive, Paper 2015/938, 2015, https://eprint.iacr.org/2015/938. [Online]. Available: https://eprint.iacr.org/2015/938

[88] [Online]. Available: https://crypto.stanford.edu/pbc/notes/crypto/ot.html

[89] "GitHub - zeromq/pyzmq: PyZMQ: Python bindings for zeromq — github.com," https://github.com/zeromq/pyzmq.

[90] "GitHub - zeromq/jeromq: Pure Java ZeroMQ — github.com," https://github.com/zeromq/jeromq.

Apéndices

A

Apéndices adicionales

A.1. Planificación detallada

A continuación, se proporcionan tablas con las horas estimadas para completar cada tarea.

Tarea	Iteración	Trabajo (horas)
Bibliografía sobre variantes de conjutnos de intersecciones privadas	1	6
Bibliografía sobre esquemas criptográficos	1	10
Bibliografía de herramientas alternativas para crear una red descentralizada	1	4
Estudio de sistemas o trabajos similares	1	5
Lectura de otros estudios realizados	1	4
Bibliografía de criptografía post-cuántica	6	6
Total estudios iniciales		**30**

Tabla A.1: Planificación de los estudios iniciales

Tarea	Iteración	Trabajo (horas)
Contexto, objetivos, alcance	A medida que se termina de fijar el alcance	2
Objetivos	1	1
Estudio de la situación actual	1	5
Planificación del proyecto	1	3
Análisis	A medida que se avanza	12
Diseño del sistema	A medida que se avanza	16
Estudio experimental y resultados	10	8
Conclusiones y trabajos futuros	10	6
Apéndices	10	3
Manuales del sistema	10	1
Colección de llamadas de la API REST	10	1
Total documentación		**68**

Tabla A.2: Planificación de la documentación

Tarea	Iteración	Trabajo (horas)
Desarrollo de interfaces	2	2
Implementación de ZMQ	2	4
Lógica de mensajería	2	2
Desarrollo del esquema criptográfico de Paillier	3	6
Lógica de intersecciones en un dominio	3	1
Lógica de intersecciones con evaluación polinómica	4	3
Desarrollo del esquema criptográfico de Damgård–Jurik	4	5
Lógica de la cardinalidad de las intersecciones con evaluación polinómica	5	2
Sistema de registros	6	8
Migración a enfoque multihilo	7	8
Adaptación del sistema de registros al multihilo	7	2
Ejecutores prioritarios	8	2
Lógica para ejecutar mútliples tareas	9	2
Total desarrollo Android		**47**

Tabla A.3: Planificación del desarrollo Android

Tarea	Iteración	Trabajo (horas)
Interfaz de la API REST	2	2
Implementación ZMQ	2	4
Lógica de mensajería	2	2
Integración de la librería phe	3	4
Lógica de intersecciones en un dominio	3	4
Lógica de intersecciones con evaluación polinómica	4	8
Integración de la librería damgard-jurik	4	5
Lógica de la cardinalidad de las intersecciones con evaluación polinómica	5	3
Sistema de registros	6	8
Migración a enfoque multihilo	7	8
Adaptación del sistema de registros al multihilo	7	5
Ejecutores prioritarios	8	5
Lógica para ejecutar múltiples tareas	9	2
Total desarrollo Python		**60**

Tabla A.4: Planificación del desarrollo en Python

Tarea	Trabajo por iteración 2-9 (horas)
Pruebas manuales y/o exploratorias	0.5
Pruebas de regresión	0.5
Pruebas integradas	0.5
Total por iteración	**1.5**

Tabla A.5: Planificación de las pruebas

Tarea	Iteración	Trabajo (horas)
Diseño de los experimentos	9	5
Ejecución de los experimentos	9	12
Análisis de los resultados	9	12
Total experimentos		**29**

Tabla A.6: Planificación de los experimentos

A.2. Requisitos no funcionales

RNF-1. Los servicios web estarán implementados usando el lenguaje de programación Python

RNF-1.1. Utilizando la versión 3.11.

RNF-2. La aplicación de Android estará implementada usando los lenguajes de programación Kotlin y Java

RNF-2.1. Usando como SDK de compilación el 34 - Android 14

RNF-2.2. Usando el Plugin de Gradle para Android versión 8.3.2

RNF-2.3 Utilizando la versión de Kotlin 2.0.0

RNF-2.4. El SDK mínimo para poder utilizar la aplicación será el 26 - Android 8.0

RNF-3. El sistema utilizará ZeroMQ [45] como librería de mensajería para implementar la red descentralizada.

RNF-3.1. Utilizando la versión 25.1.2 de pyzmq [89] en Python.

RNF-3.2. Utilizando la versión 0.6.0 de jeromq [90] para el desarrollo Android.

RNF-3.3. Soportará IPv4 e IPv6.

RNF-3.4. Los mensajes se enviarán mediante sockets *DEALER* a las partes interesadas.

RNF-3.5. La información sensible (los conjuntos de datos) se enviarán siempre de forma cifrada.

RNF-4. Se utilizará la Firebase Realtime Database [52] como base de datos para los registros generados, se registrarán las actividades de generación de claves, los resultados, las configuraciones y una traza de tiempo y recursos utilizados por cada operación de intersección, con un código de actividad asociado. Su acceso de escritura estará limitado por autenticación.

RNF-4.1. Los dispositivos escribirán bajo el nodo "logs".

RNF-4.2. Los dispositivos usarán su dirección IP como identificador en la base de datos.

RNF-4.3. Los dispositivos Android usarán el SDK oficial de Google y un usuario autenticado previamente configurado desde la consola de Firebase. Por seguridad, el archivo de propiedades que incluya este usuario estará en los dispositivos que compilen el *APK*, y no será público en el repositorio. Por motivos de privacidad, habrá dos compilaciones, una con el archivo, y otra sin la capacidad de mandar registros.

RNF-4.4. Los servicios web usarán la librería: *Firebase Admin Python SDK* para realizar las operaciones CRUD. Versión 6.5.0. El archivo de credenciales estará en los dispositivos que corran el servicio y no estará en el repositorio.

RNF-4.5. El sistema podrá funcionar sin autenticación, en este modo, no enviará registros.

RNF-5. Ninguno de los dos sistemas será persistente, un nodo nuevo será creado cada vez que se inicie.

RNF-6. Los dispositivos Android reportarán a Firebase Crashlytics [53] en caso de tener

algún error.

RNF-7. Los servicios web harán uso de Flask [43] para crear el servicio y la API REST. Versión 3.0.2.

RNF-8. Los servicios web usarán la librería "phe-[37] como implementación del esquema criptográfico de Paillier. Versión 1.5.0.

RNF-9. Los servicios web usarán la implementación "damgard-jurik-[38] como esquema criptográfico de Damgård-Jurik. Versión 0.0.3.

RNF-10. la comunicación con la API REST desarrollada en el servicio web se realizará mediante el protocolo HTTP.

RNF-11. Los dispositivos deberánn desencadenar acciones automáticas en base a mensajes recibidos.

RNF-12. El nodo de red se debe inicializar automáticamente cuando se inicie el sistema, pudiendo modificar su estado posteriormente.

RNF-13. La consola de Python y el Logcat de Android darán información valiosa con respecto a las operaciones que se están realizando.

RNF-14. La interfaz de Android y el *frontend* de Python deben ser accesibles, intuitivos y fáciles de usar, incluso para usuarios sin experiencia técnica.

RNF-15. Toda posible interacción que pueda resultar en algo no conocido para el sistema debe desencadenar un error no bloqueante, para poder seguir utilizando el sistema posteriormente.

RFN-16. La función de ping lanzará 3 peticiones espaciadas en el tiempo para comprobar si un usuario está conectado. Se considerará que un usuario está desconectado si después de esos intentos no se ha recibido una respuesta. Estas respuestas las tiene que mandar el socket *ROUTER*.

RFN-17. El sistema debe ser capaz de procesar mensajes y calcular intersecciones simultáneamente, priorizando otros mensajes (como *ping* o *discover*) antes que los *JSON*, o los mensajes recibidos por encima de los que se le han mandado procesar.

RNF-18. Se podrán inicializar los nodos con compañeros por defecto, a los que intentará contactar durante su instanciación.

RNF-19. El sistema debe mostrar en las interfaces (y mediante petición en la API REST) el número de acciones que le quedan pendientes. Las interfaces actualizarán este valor cada segundo.

A.3. Casos de Uso detallados

Tabla A.7: Casos de uso detallados

CU-001	Inicializar nodo
Actor	Usuario desconectado
Descripción	

El usuario iniciará manualmente el nodo indicando el puerto de escucha, bien mediante la opción de Android, el *frontend* del servicio web, o la API REST. El sistema comprueba que no hay otro nodo ya creado y que puede obtener el identificador de red. El sistema inicia el nodo y este envía la información de su configuración a la base de datos si esta activada y autenticada.	
CU-002	Añadir compañeros
Actor	Usuario conectado
Descripción	
El usuario utiliza la función de descubrimiento para área local, o añade manualmente un dispositivo usando su identificador (IP,), independientemente de si está en la red local o no.	
CU-003	Detección de nuevos dispositivos
Actor	Usuario conectado
Descripción	
El sistema recibe un mensaje de un dispositivo y lo consume. Si este dispositivo no está en su lista de dispositivos, lo añade siguiendo el **CU-002**. El usuario puede ver el nuevo dispositivo en la aplicación, refrescando el *frontend* y consultando la API REST.	
CU-004	Generación manual de claves
Actor	Usuario conectado
Descripción	
El usuario utiliza cualquiera de los sistemas para generar nuevas claves de los esquemas criptográficos implementados. Mediante la aplicación y la API, se debe indicar la longitud de claves a generar. El sistema comprobará que el tamaño de la clave es válido y realizará la función en segundo plano. Mandará un registro a la base de datos con los recursos utilizados en dicha operación si esta está activada y autenticada.	
CU-005	Iniciar cálculo de PSI
Actor	Usuario conectado
Descripción	
El usuario elige un dispositivo de la red, el esquema criptográfico que quiere usar y la técnica que quiere aplicar. El sistema comprueba que dicho dispositivo está en la lista de dispositivos y serializa los parámetros necesarios acordemente, envía un JSON al usuario seleccionado utilizando su socket. Esto lo realiza en segundo plano, sin bloquear al usuario para realizar otras operaciones. Mandará un registro a la base de datos con los recursos utilizados y el código de la operación si está activada y autenticada.	
CU-005.1	Desde la aplicación de Android y la API puede seleccionar el número de veces que quiere iniciar el cálculo.
CU-006	Lanzar *tests*
Actor	Usuario conectado
Descripción	
El usuario selecciona la opción de *test* por defecto contra un dispositivo. El sistema comprueba que el dispositivo existe, y comienza el proceso análogamente a cómo se realiza en el **CU-005** las veces que se le haya solicitado.	
CU-007	Acceder a los resultados
Actor	Usuario conectado
Descripción	
El usuario accede bien a la pantalla de resultados utilizando el *frontend* o la aplicación, o llama a la API REST para que le de los resultados que ha registrado el nodo en el tiempo que ha estado activo. Los datos se muestran como un JSON en los servicios web, y cómo cadenas de texto en Android.	
CU-008	Ver las claves públicas
Actor	Usuario conectado

Descripción	
El usuario accede bien a la pantalla de claves utilizando el *frontend* o la aplicación, o llama a la API REST para que le proporcione las claves públicas de los criptosistemas implementados. Se devuelve un JSON en los servicios web y cadenas de texto en Android.	
CU-009	Ver los datos del nodo
Actor	Usuario conectado
Descripción	
El usuario accede bien a la pantalla de claves utilizando el *frontend* o la aplicación, o llama a la API REST para conocer el conjunto de datos que está enviando cuando inicia los procesos del **CU-005**. Se devuelve un JSON en los servicios web, y una cadena de texto en Android.	
CU-010	Apagar el nodo
Actor	Usuario conectado
Descripción	
El usuario selecciona en la interfaz la opción de desconexión o llama a la API para apagar el nodo que está corriendo. El sistema cierra todos los sockets y destruye la instancia de nodo actual.	
CU-011	Actualizar datos
Actor	Usuario conectado
Descripción	
El usuario indica el nuevo tamaño del set y el dominio sobre el que opera. El sistema comprueba que el tamaño del dominio sea mayor que el set y genera los nuevos datos aleatorios. Guarda la configuración en Firebase si está autenticado.	
CU-012	Activar o desactivar envío de registros
Actor	Todos
Descripción	
El usuario indica si quiere activar o desactivar el envío de registros a Firebase mediante la opción en la aplicación o una llamada a la API. El sistema comprobará si la compilación o distribución dispone del archivo de credenciales (Python) o propiedades (Android) para poder realizar la función.	

A.4. Diagrama de clases de red y criptografía - Servicios Web

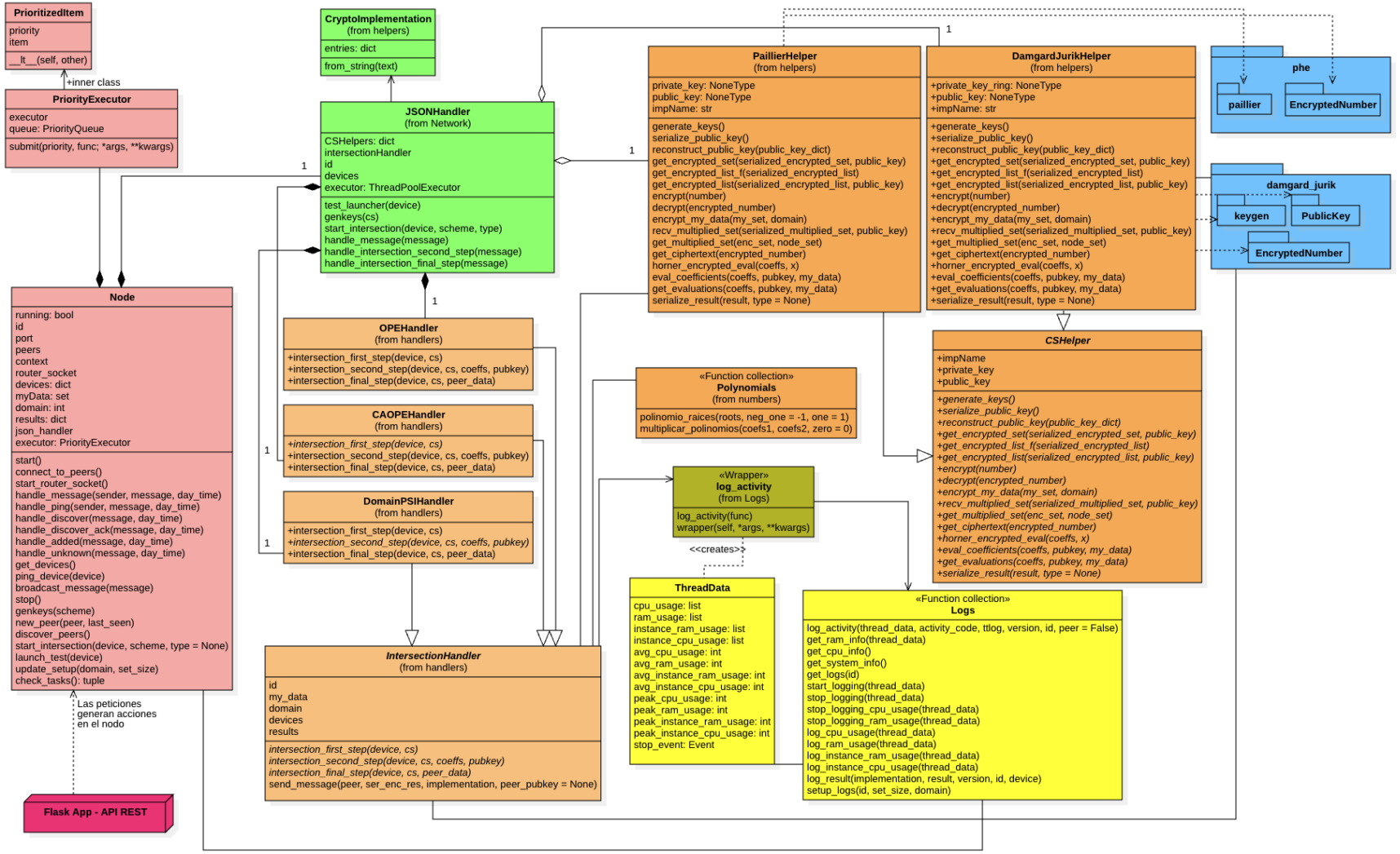

Figura A.1: Diagrama de clases de red y criptografía - Python

A.5. Tiempos de cómputo con claves de 1024 y 512 bits

Claves de 1024 bits

Criptosistema	Protocolo	Paso 1 (WS)	Paso 2 (Android)	Paso 3 (WS)	Total
Paillier	Según el dominio	4,1686	1,1619	1,3793	6,7098
	OPE	0,2612	1,3570	0,1658	1,7841
	OPE - c	0,2564	1,3492	0,1661	1,7718
Damgård-Jurik	Según el dominio	26,4271	2,8543	25,4143	54,6957
	OPE	2,3104	1,9678	2,5404	6,8186
	OPE-C	2,2173	3,2470	2,6143	8,0788

Tabla A.8: Servicio web inicia el proceso de intersección contra un dispositivo Android. Resultados - Claves de 1024 bits - Tiempo en segundos

Criptosistema	Protocolo	Paso 1 (Android)	Paso 2 (WS)	Paso 3 (Android)	Total
Paillier	Según el dominio	2,5893	2,3124	4,9867	9,8885
	OPE	0,3760	1,2335	0,4084	2,0179
	OPE - c	0,3301	1,19427	0,2697	1,7940
Damgård-Jurik	Según el dominio	12,6935	15,2950	6,6217	34,6103
	OPE	1,5494	2,5551	0,5532	4,6578
	OPE - c	1,5444	2,5844	0,5473	4,6761

Tabla A.9: Dispositivo Android inicia el proceso de intersección contra un servicio web. Resultados - Claves de 1024 bits - Tiempo en segundos

Claves de 512 bits

Criptosistema	Protocolo	Paso 1 (WS)	Paso 2 (Android)	Paso 3 (WS)	Total
Paillier	Según el dominio	0,7755	0,3144	0,2086	1,2985
	OPE	0,0410	0,5713	0,0680	0,6803
	OPE - c	0,0357	0,5601	0,0664	0,6622
Damgård-Jurik	Según el dominio	3,0597	0,8515	3,1538	7,0650
	OPE	0,21748	0,8722	0,3120	1,4017
	OPE - c	0,2212	0,8582	0,30745	1,3868

Tabla A.10: Servicio web inicia el proceso de intersección contra un dispositivo Android. Resultados - Claves de 512 bits - Tiempo en segundos

Criptosistema	Protocolo	Paso 1 (Android)	Paso 2 (WS)	Paso 3 (Android)	Total
Paillier	Según el dominio	0,4062	0,7206	0,3031	1,4300
	OPE	0,0629	0,3070	0,0232	0,3931
	OPE - c	0,0629	0,2888	0,0209	0,3725
Damgård-Jurik	Según el dominio	1,5280	2,2469	0,6106	4,3855
	OPE	0,2374	0,3889	0,0837	0,7100
	OPE - c	0,2772	0,3858	0,0770	0,7401

Tabla A.11: Dispositivo Android inicia el proceso de intersección contra un servicio web. Resultados - Claves de 512 bits - Tiempo en segundos