Universidad de Cantabria

Facultad de Ciencias

Estudio e Implementación de un Esquema de PSI (Private Set Intersection) Descentralizado

Study and Implementation of a Decentralized PSI (Private Set Intersection) Scheme

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jiale Zhang

Director: Domingo Gómez Pérez

Noviembre de 2024

Agradecimientos

Quiero expresar mi agradecimiento a mi mentor, Domingo Gómez Pérez, por su invaluable orientación, paciencia y dedicación durante el desarrollo de este TFG. Su guianza y acertadas recomendaciones han sido claves para el éxito de este proyecto.
Este proyecto se ha realizado dentro de las actividades de la CÁTEDRA UNIVERSIDAD DE CANTABRIA-INCIBE DE NUEVOS RETOS EN CIBERSEGURIDAD. Financiado por la Unión Europea NextGeneration-EU, Plan de Recuperación, Transformación y Resiliencia, a través de INCIBE.

Estudio e Implementación de un Esquema de PSI (Private Set Intersection) Descentralizado

Resumen

palabras clave: PSI, Criptosistema, Paillier, Damgard-Jurik

El matemático estadounidense Leonard Dickson (1874-1954) expresó en su época: "Thank God that number theory is unsullied by any application". Sin embargo, lo que Dickson no pudo prever en su momento es que con la llegada de las computadoras, la teoría de números encontraría aplicaciones de gran relevancia en dicho campo. Una de las áreas donde está se manifiesta de manera notable es en el campo de la criptografía, donde la teoría de números juega un papel fundamental en el diseño y la seguridad de los sistemas criptográficos.

Uno de estos esquemas, los PSI (Private Set Intersection), son sistemas criptográficos de gran relevancia en la actualidad. Estos hacen uso de técnicas de encriptación homomórfica para abordar un problema emergente: la necesidad de que dos entidades sin confianza mutua puedan encontrar la intersección de dos conjuntos de datos sin exponer información adicional. Este tipo de técnicas encuentra aplicación en campos como la publicidad, medicina o el descubrimiento de contactos mutuos sin revelar información sensible. Recientemente, se han propuesto extensiones de estos protocolos para múltiples entidades, como grupos de usuarios, en una red social segura o en sistemas de mensajería.

En este trabajo, se estudiará la implementación de sistemas clásicos con criptosistema homomórficos como el de Paillier, así como propuestas más recientes como el de Damgard-Jurik. Se comparará su rendimiento utilizando métricas como el tamaño de los mensajes para un nivel de seguridad dado, así como el tamaño de las claves definidas.

Study and Implementation of a Decentralized PSI (Private Set Intersection) Scheme

Abstract

keywords: PSI, Criptosystem, Paillier, Damgard-Jurik

The American mathematician Leonard Dickson (1874-1954) expressed in his time: "Thank God that number theory is unsullied by any application."However, what Dickson could not foresee at the time is that with the arrival of computers, number theory would find highly relevant applications in this field. One of the areas where this manifests itself notably is in the field of cryptography, where number theory plays a fundamental role in the design and security of cryptographic systems.

One of these schemes, PSI (Private Set Intersection), are cryptographic systems of great relevance today. These make use of homomorphic encryption techniques to address an emerging problem: the need for two entities without mutual trust to be able to find the intersection of two data sets without exposing additional information. This type of technique finds application in fields such as advertising, medicine or the discovery of mutual contacts without revealing sensitive information. Recently, extensions of these protocols have been proposed for multiple entities, such as groups of users in a secure social network or in messaging systems.

In this work, the implementation of classical systems using homomorphic cryptosystems such as Paillier will be studied, as well as more recent proposals such as Damgard-Jurik. Their performance will be compared using metrics such as message size for a given security level, as well as the size of the defined keys.

Índice

1 Introducción

En un mundo cada vez más interconectado, en el que el intercambio de información entre entidades es constante, la seguridad y privacidad de dichas comunicaciones es de suma relevancia. Mediante la criptografía se aborda este desafío, permitiendo proteger la confidencialidad de los individuos, la integridad y la autenticidad de los datos en tránsito entre diversas organizaciones e individuos.

Entre las dificultades encontradas durante el intercambio seguro de información existe un área de especial interés relacionado con la problemática de permitir que un grupo de entidades independientes computen de forma conjunta operaciones sobre sus datos privados de forma segura, asegurando que cada participante solo acceda al resultado final, sin exponer su información individual. La familia de protocolos que resuelven estos problemas son denominados como Computación Segura Multipartito (Secure Multiparty Computation, SMPC).

Un caso particular de este conjunto de protocolos es el PSI (Private Set Intersection), una técnica que permite a dos o más entidades calcular la intersección entre sus conjuntos de datos privados. A partir del PSI, las entidades pueden identificar elementos comunes entre sus conjuntos, mientras que los elementos que no se encuentran en el mismo permanecen ocultos.

Actualmente, el PSI ya se ha implementado y utilizado en diversas situaciones de la vida cotidiana. Por ejemplo, en la detección de fraudes en sistemas financieros, diferentes entidades pueden verificar si un cliente muestra actividades sospechosas comparando transacciones sin revelar información adicional confidencial. En el ámbito de la salud, PSI permite a hospitales e instituciones compartir datos sobre síntomas y enfermedades para investigaciones sin comprometer la privacidad de los pacientes.

En redes sociales, PSI facilita la migración entre plataformas sociales y compartir contenido entre usuarios. Por ejemplo, Signal es una aplicación de mensajería que utiliza PSI para permitir a los usuarios descubrir cuáles de sus contactos están en la plataforma, sin que el servidor de Signal obtenga información sobre las listas de contactos privadas de los usuarios (Signal Foundation [2018]).

Por otro lado, existen otras aplicaciones como AirDrop (de Apple), que permiten el intercambio de archivos entre usuarios de manera aparentemente anónima, permitiendo configurar la opción de aceptar documentos de “solo contactos” o de “todo el mundo”. No obstante, el protocolo que emplea Apple presenta vulnerabilidades importantes para la privacidad (Green [2024]). Esto es debido a que antes de iniciar la comunicación, el dispositivo emisor envía un *hash* del número de teléfono y dirección de correo del usuario por la red. Debido al reducido dominio de estos datos, un atacante puede realizar un “dictionary attack” que le permita obtener la información de contacto del usuario de origen.

Esta vulnerabilidad podría mitigarse con la implementación de protocolos de PSI. Esto garantizaría que el proceso de descubrimiento fuese completamente privado, preservando la identidad de los usuarios y eliminando el riesgo de rastreo. Sin embargo, su uso en AirDrop enfrenta barreras políticas y prácticas porque en regiones como China, esta tecnología dificultaría los controles gubernamentales, ya que AirDrop se utiliza con frecuencia para compartir archivos censurados.

Como se puede observar, la Intersección Privada de Conjuntos (PSI) tiene aplicaciones relevantes en la actualidad. Además, es una técnica que ha evolucionado considerablemente desde la primera versión introducida. En consecuencia, existen diferentes implementaciones de PSI, cada una empleando diversas primitivas criptográficas para su construcción. En este documento, (Morales et al. [2023]) presenta una taxonomía y clasificación de los protocolos PSI disponibles. Para definir el alcance de

este proyecto, nos centraremos en las implementaciones que usan la criptografía homomórfica como principal primitiva para su construcción.

Para facilitar la comprensión al lector, se presentarán una serie de conceptos clave que se deberán tener en cuenta a lo largo del trabajo. Es importante recordar que los ordenadores procesan y almacenan información en forma de bytes. Esto significa que no transmiten ni almacenan texto de forma directa, sino una secuencia de unos y ceros (1's y 0's) que adquiere significado solo al aplicar un estándar de codificación adecuado.

Para ello, se utiliza Unicode, un estándar de codificación de caracteres diseñado para representar texto en la mayoría de los sistemas de escritura del mundo. Antes de su adopción, existían numerosos estándares diferentes de codificación que usaban 8 bits, como ASCII (American Standard Code for Information Interchange), EBCDIC y codificaciones locales dadas por el ISO 8859. Sin embargo, estos sistemas tenían limitaciones significativas en cuanto a la cantidad de caracteres que podían representar, especialmente para idiomas y escrituras no latinas, ya que solo permitían un máximo de 256 combinaciones distintas.

Unicode soluciona esta limitación al proporcionar un conjunto de hasta 1,114,112 combinaciones posibles, asignando un número único, normalmente representado en hexadecimal, a cada carácter. Esto permite representar casi todos los caracteres utilizados en la comunicación escrita. Esto incluye caracteres de idiomas, símbolos matemáticos, emoticonos y mucho más. Para su almacenamiento en bytes, Unicode ofrece diferentes formatos de transformación, o Unicode Transformation Formats (UTF). Por ejemplo, UTF-8 es una codificación muy utilizada, ya que emplea una longitud variable de entre 1 y 4 bytes para representar cada carácter, lo que la hace eficiente en términos de espacio.

En general, cuando se aplica un algoritmo de encriptación sobre un mensaje o documento, este suele estar codificado en Unicode. El texto resultante, tras la encriptación, se convierte en una serie de símbolos y caracteres sin significado alguno para un ser humano, asegurando así la confidencialidad de la información.

1.1. Teoría de la Complejidad Computacional

No obstante, la seguridad en la criptografía no solo depende de cómo se representan los datos, como en el caso del estándar Unicode, sino también de la dificultad inherente a resolver ciertos problemas matemáticos. Es aquí donde entra en juego la Teoría de la Complejidad Computacional, una disciplina que se centra en clasificar problemas computacionales en función de la cantidad de recursos espaciales y temporales que necesita un computador parar resolver un determinado problema mediante un algoritmo.

Para la criptografía, la teoría de la complejidad es de gran importancia, ya que la seguridad de muchos sistemas criptográficos se basa en la dificultad de resolver ciertos problemas matemáticos en un tiempo razonable. Existen diferentes tipos de problemas computacionales, como los problemas de decisión o los problemas de búsqueda, siendo uno de los más relevantes en la criptografía la factorización de enteros, en cuya complejidad se basan ciertos sistemas criptográficos muy conocidos como el de RSA (Rivest et al. [1978]).

La complejidad de un problema se mide en función del tamaño de la entrada, generalmente representado en bits. Por ejemplo, si el problema consiste en operar con números enteros, el tamaño de la entrada está dado por la cantidad de bits necesarios para representar dichos números. Así, un problema de entrada x tiene un tamaño n, donde $n = \lceil \log_2(x) \rceil$.

La complejidad temporal, o tiempo de ejecución, de un algoritmo se expresa en función del tamaño de entrada n. Por otro lado, la complejidad espacial se refiere a la cantidad de memoria necesaria para resolver el problema.

Teniendo esto en cuenta, un problema es considerado complicado si los recursos necesarios para resolverlo son excesivos, es decir, si su resolución práctica no es factible debido a las restricciones

temporales o espaciales.

Esta noción de complejidad se formaliza mediante modelos matemáticos de computación, como las máquinas de Turing, que permiten clasificar los problemas según la cantidad de recursos requeridos. Estos modelos son fundamentales para evaluar la viabilidad de los sistemas criptográficos basados en problemas complejos.

Dentro de la teoría de la complejidad, uno de los conceptos clave es la intratabilidad. Un problema se denomina intratable cuando, aunque teóricamente es resoluble, el coste computacional necesario para encontrar una solución viable es tan elevado que lo convierte en inaplicable en escenarios prácticos.

1.2. Metas y Alcance del Proyecto

En esta sección se establecen los objetivos y el alcance del proyecto, además de exponer la metodología y el modelo empleado para el desarrollo del proyecto.

En primer lugar, se estudiaron y analizaron los requisitos del proyecto mediante una entrevista con el tutor. El objetivo del trabajo es implementar los criptosistemas de Paillier y Damgard-Jurik y utilizar dichas primitivas para implementar protocolos diferentes de PSI. A partir del código desarrollado se busca analizar el rendimiento temporal de cada implementación mediante diferentes métricas.

Entre las técnicas a analizar se han escogido tres métodos que emplean la criptografía homomórfica como principal primitiva, cada una basada en un artículo:

- Oblivious Polynomial Evaluation PSI (Freedman et al. [2004]). El sistema se basa en que el propietario de un conjunto convierte sus elementos en raíces de un polinomio $P(x)$ y lo cifra utilizando un esquema de encriptación homomórfica. Luego, el otro participante evalúa el polinomio cifrado con sus propios elementos como entradas. Si un elemento pertenece al conjunto original, la evaluación del polinomio dará como resultado cero. Su principal ventaja es que reduce significativamente las rondas de comunicación en comparación con otros protocolos. Sin embargo, puede ser computacionalmente costoso debido a la necesidad de realizar operaciones sobre datos cifrados.

- Bloom Filter PSI (Wang et al. [2018]). Esta técnica utiliza Bloom Filters, estructuras de datos compactas y probabilísticas que permiten representar conjuntos de manera eficiente. Cada participante codifica su conjunto en un Bloom Filter, que luego se comparte o se compara con el del otro participante. Los Bloom Filters permiten determinar la intersección de los conjuntos utilizando operaciones lógicas (como AND). No obstante, hay una pequeña probabilidad de falsos positivos, debido a las propiedades de estas estructuras de datos.

- Bit Vector PSI (Ruan et al. [2019]). En esta técnica, los conjuntos se codifican como vectores binarios (bit vectors), donde cada bit representa la presencia o ausencia de un elemento en un dominio predefinido. Ambos participantes generan vectores de bits correspondientes a sus conjuntos y realizan operaciones bit a bit para encontrar la intersección. Sin embargo, este modelo no es escalable para conjuntos con dominios extremadamente grandes, ya que los vectores de bits crecen proporcionalmente con el tamaño del dominio.

Existen diversos protocolos adicionales que no se han analizado, como los basados en *Commutative Encryption*, que es un sistema de cifrado en el orden de las operaciones de cifrado no afecta al resultado final, permitiendo la intersección de conjuntos sin revelar información. La idea básica es que un cliente C encripta sus datos con su clave pública y los envía al servidor S. El servidor cifra los datos encriptados y su conjunto de datos privados con su propia clave pública y los envía de vuelta al cliente. Al ser encriptación conmutativa, el cliente puede desencriptar sus datos encriptados por el servidor y compararlo con el conjunto encriptado de datos del servidor, pudiendo encontrar la intersección sin revelar información adicional.

También están aquellos sistemas basados en Pairings, los cuales aprovechan propiedades matemáticas de emparejamientos bilineales en grupos algebraicos para garantizar la seguridad y privacidad en la comparación de conjuntos.

Además, hay extensiones del protocolo que no se han abarcado, como el PSI-CA, donde solo se revela la cardinalidad de la intersección, así como el MP-PSI, que permite que más de dos entidades computen su intersección de manera segura.

Con el fin de realizar las pruebas y recopilar datos, se ha desarrollado una sencilla interfaz gráfica mediante *Flask*, la cual expone una API a la que se pueden realizar peticiones. A través de esta interfaz, se pueden cargar grupos de datos de diferentes tamaños, además de generar conjuntos de manera aleatoria, especificando el tamaño máximo del conjunto y el valor máximo de los elementos que lo componen. Asimismo, es posible seleccionar el esquema de clave pública a emplear, así como el tamaño de dicha clave. En el caso de Damgård-Jurik, se ofrece la opción de configurar el índice de expansión (s). Adicionalmente, se puede elegir el protocolo PSI a utilizar de los que se han implementado.

Finalmente, los datos temporales obtenidos de las diferentes etapas del PSI, se presentan en formato tabular en una página separada, con la opción de exportar dicha tabla a un archivo .csv para su facilitar posterior análisis y la creación de gráficos.

Para el desarrollo y la orientación del proyecto, han sido de gran ayuda los apuntes proporcionados por el tutor sobre la Intersección Privada de Conjuntos, los cuales me han permitido comprender el funcionamiento de los criptosistemas implementados y entender los conceptos matemáticos relacionados. Es debido a ello que la estructura de este trabajo se ha dividido en los siguientes 3 capítulos.

En el Capítulo 2 nos centramos en introducir una base matemática que permita entender y asimilar el funcionamiento de los criptosistemas homomórficos a emplear. En este apartado se abordan conceptos matemáticos fundamentales, como que es un grupo, un anillo, los números primos y las congruencias, son introducidos en este apartado, además de la explicación de los criptosistemas de Paillier y Damgard-Jurik.

El Capítulo 3 se explican de detalladamente la organización y los conceptos detrás delos tres diferentes protocolos que se han implementado, así como las peculiaridades de cada uno. Nos centramos en el funcionamiento y los pasos de dichas técnicas, así como sus diferentes casos de uso.

Finalmente, en el Capítulo 4 exponemos el proceso seguido para la implementación de los esquemas de clave pública y los diferentes protocolos PSI implementados, así como el desarrollo de la aplicación. En esta sección también se muestran y analizan los resultados obtenidos de las pruebas realizadas, teniendo en cuenta distintas métricas como el tamaño de la clave o el número de elementos de los conjuntos de entrada.

2 Base Matemática

En este capítulo recopilamos los principales conceptos matemáticos necesarios para entender los criptosistemas de Paillier y Damgard-Jurik y la consecuente construcción del esquema PSI. En general, se expondrán conceptos básicos de Teoría de Números y Álgebra Abstracta, sin entrar demasiado en detalle, solo lo justo para comprender mejor el desarrollo de los capítulos 3 y 4. La mayoría del material empleado para desarrollar esta sección se han sacado de Robinson [2003]; Rosen [2011].

2.1. Enteros y Números Primos

Los números primos son un concepto fundamental en teoría de números y juegan un papel crucial en muchas áreas de la criptografía moderna. En esta sección, se presentarán las propiedades básicas de los números primos y su relevancia en el ámbito criptográfico.

Definición 1 (Número Primo)**.** *Un número entero $p > 1$ es considerado primo si tiene exactamente dos divisores positivos (1 y p). El número 1 no se considera primo.*

Definición 2 (Número Compuesto)**.** *Un número entero n mayor que 1 que no es primo se denomina número compuesto. Un número compuesto tiene más de dos divisores positivos. Por ejemplo, 6 es compuesto porque sus divisores son $1, 2, 3, 6$.*

Algunas de las propiedades más interesantes de los números primos para la criptografía son:

- Los números primos son los 'bloques' de los que están conformados los números enteros:

  Teorema 1 ((Teorema Fundamental de la Aritmética))**.** *Todo número entero mayor que 1 puede expresarse de manera única como un producto de números primos.*

  Como ejemplo, $30 = 2 \cdot 3 \cdot 5$. Aunque se sabe que existe y es única, la factorización es un problema difícil, que hace que sea utilizado en criptografia.

- Los números primos son utilizados en muchas técnicas criptográficas, como RSA, que basan su intratabilidad en el problema de factorización de números enteros de gran tamaño, afirmando la seguridad del criptosistema.

- En criptografía, se utilizan a menudo números primos para definir grupos cíclicos y crear sistemas de clave pública. La estructura algebraica que proporciona un número primo p ayuda a asegurar propiedades como la inyectividad y la eficiencia computacional.

Son de suma importancia los algoritmos para encontrar los números primos menores que un número dado n. Uno de los primeros algoritmos conocidos es el de la **Criba de Eratóstenes**, que funciona de la siguiente manera:

Ejemplo 1 (Criba de Erastótenes)**.** *El algoritmo que sigue este método para determinar la primacidad de un número entero cualquiera n es el siguiente:*

1. *Se crea una lista de todos los números desde 2 a n.*

2. *Se eliminan todos los múltiplos del primer número primo (2) de la lista.*

3. *Luego se pasa al siguiente número no eliminado de la lista (el siguiente primo) y se eliminan sus múltiplos.*

4. *Este proceso continúa hasta que se han eliminado todos los múltiplos de cada número primo hasta* $\sqrt{n}$.

Al final, los números no marcados en la lista son los números primos.

Claramente, este proceso es muy ineficiente y para números grandes el tiempo de cómputo es inmenso. No obstante, se han ido desarrollando algoritmos más eficientes con el paso del tiempo. Debido a ello, en la criptografía moderna, se utilizan **primos grandes** (con cientos o miles de dígitos) para asegurar la dificultad computacional de los algoritmos criptográficos. El tamaño de los números primos utilizados en sistemas como RSA es un factor clave que asegura la robustez de estos sistemas frente a ataques de fuerza bruta y otras técnicas de factorización.

2.2. Grupos y Anillos

Los conceptos matemáticos de grupos y anillos son importantes para definir, analizar y entender los sistemas criptográficos. Por ejemplo, sirven para entender el problema del logaritmo discreto, un problema matemático fundamental en el campo de la criptografía.

Definición 3 (Logaritmo discreto)**.** *El problema del logaritmo discreto es, dados* g, h, p *números enteros, encontrar un número entero* x *tal que:*

$$g^x = h \mod p,$$

donde p *es un primo.*

Observación 1. *El problema del logaritmo discreto se le llama función de un solo sentido porque a pesar de que obtener* h *a partir de* g *y* x *es fácil, no se conoce como obtener* x *de forma eficiente.*

Este problema es un problema más general que se plantea usando *Teoría de Grupos*. Un grupo se define como un conjunto de elementos con una operación que puede ser aplicada a dichos elementos. Un grupo también cuenta con una serie propiedades que se tienen que satisfacer. Se podría decir que la noción de grupo nace de una generalización de la operación suma sobre el conjunto de los números enteros.

Por ejemplo, el conjunto de los números enteros es un grupo con la suma. Además, dado un número entero x, si se le aplica la operación suma con el 0, sigue siendo x, que en el contexto de la teoría de grupos, se condensa en la existencia de un elemento neutro para la operación considerada. La idea de anillo surge de forma natural cuando un grupo además posee otra operación como el concepto de la multiplicación en los números enteros.

Naturalmente, las nociones de grupo y anillo son mucho más amplias, por lo que las operaciones consideradas no tienen por qué corresponderse con la suma y multiplicación de enteros.

Definición 4. *Un grupo* $(G, \cdot)$ *es un conjunto* G*, dotado de una aplicación* $\cdot : G \times G \to G$ *que denominaremos la operación del grupo* G*, la cual satisface las siguientes propiedades:*

1. ***Asociatividad:*** *Para todo* $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

2. ***Elemento neutro:*** *Existe un elemento* $e \in G$ *tal que para todo* $a \in G$, $a \cdot e = e \cdot a = a$.

3. ***Inverso:*** *Para cada elemento* $a \in G$*, existe un elemento* $a^{-1} \in G$ *tal que* $a \cdot a^{-1} = a^{-1} \cdot a = e$.

El problema del logaritmo discreto se podría reinterpretar como que estamos trabajando en un grupo finito cíclico generado por g.

Observación 2. *En lo que sigue, realizaremos un abuso de notación y abreviaremos el grupo $(G, \cdot)$ como G si no existe riesgo de confusión sobre la operación considerada.*

Cabe destacar que una partición de elementos de G, llamémoslo H, se considera subgrupo de G si con la misma operación $\cdot$ y elemento neutro e, $(H, \cdot)$ es un grupo. Un teorema relevante para entender los criptosistemas de Paillier y Damgard-Jurik que se van a implementar posteriormente es el siguiente.

Teorema 2 ((Teorema de Lagrange))**.** *Sea G un grupo finito y H un subgrupo de G. Entonces, el orden (el número de elementos) de H divide al orden de G.*

Además, el orden de un elemento $x \in G$ divide también el orden de G.

Observación 3. *El orden de un grupo G es el número de elementos de dicho grupo. Mientras que el orden de un elemento g de un grupo G es el número de veces que se debe aplicar la operación del grupo a g para obtener el elemento neutro. Es decir, el orden de g, denotado como $|g|$ es el menor n para el que se cumple:*

$$g^n = e$$

Siendo e el elemento neutro del grupo.

Una vez explicados estos conceptos, podemos introducir el grupo multiplicativo de enteros módulo n, fundamental en la criptografía. Este se denota típicamente como $(\mathbb{Z}_n)^\times$, y está compuesto por todos los números enteros que son coprimos con n, es decir, aquellos cuya único divisor común con n es 1.

Este grupo tiene propiedades interesantes para la construcción de sistemas criptográficos. En primer lugar, es un grupo abeliano, lo que significa que para cualquier par de elementos $a, b \in (\mathbb{Z}_n)^\times$, se cumple que $a \cdot b = b \cdot a$. Además, el grupo es finito, y su orden está dado por la función de Euler $\phi(n)$, que devuelve la cantidad de números coprimos con n y menores que él. Por lo tanto, el orden del grupo $(\mathbb{Z}_n)^\times$ es precisamente $\phi(n)$.

Cuando n es un número primo, el grupo $(\mathbb{Z}_n)^\times$ tiene una propiedad adicional: es cíclico, es decir, existe un elemento $g \in (\mathbb{Z}_n)^\times$ tal que el conjunto $\{g^k \mid k \in \mathbb{Z}\}$ genera todos los elementos del grupo. Además, dado que n es primo, el orden del grupo es simplemente $n-1$, como se deduce de la definición de $\phi(n)$.

En criptografía, estas propiedades son fundamentales. Por ejemplo, en el criptosistema de Paillier, los mensajes sin cifrar pertenecen al anillo de clases de restos $\mathbb{Z}_n$, mientras que los mensajes cifrados residen en el anillo $\mathbb{Z}_{n^2}$. Este esquema aprovecha la estructura del grupo multiplicativo para garantizar la seguridad y funcionalidad del sistema.

2.2.1. Las transformaciones de los grupos: Morfismos

Tras haber introducido el concepto de grupo, se presenta ahora el concepto de morfismos. Los morfismos permiten representar problemas en otro espacio donde, a menudo, resultan más sencillos de resolver:

Definición 5 (Morfismo de grupos)**.** *Sean $(G, \cdot), (H, *)$ dos grupos. Un morfismo de grupos de G a H es una aplicación $f : G \to H$ tal que $f(e_G) = e_H$ y $f(g_1 \cdot g_2) = f(g_1) * f(g_2)$. Es decir, la imagen del elemento neutro de G es el elemento neutro de H y preserva las operaciones de cada grupo.*

Un morfismo de grupos se dice que es un isomorfismo si es biyectivo. En este caso, G y H se dicen isomorfimos (es decir, equivalentes).

Un morfismo de grupos es inyectivo si su núcleo

$$\ker(f) := \{g \in G : f(g) = e_H\} = \{e_G\}$$

es decir, si el único elemento que tiene como imagen el elemento neutro de H es el elemento neutro de G.

Ejemplo 2 (Codificación UTF-8 como morfismo)**.** *Consideremos el conjunto de caracteres Unicode ($\mathcal{U}$), que incluye una amplia gama de símbolos empleados en la informática, y el formato UTF-8, $\mathcal{B}$ como sus representaciones en bytes.*

En este contexto, la codificación UTF-8 puede interpretarse como una transformación entre los caracteres en $\mathcal{U}$ y sus representaciones en bytes en $\mathcal{B}$. A cada carácter de texto en $\mathcal{U}$ le corresponde una secuencia única de bytes en UTF-8. Además, la operación "concatenación de elementos" se preserva entre grupos. Esta transformación es crucial para representar texto en sistemas informáticos, ya que convierte caracteres abstractos en secuencias binarias entendibles por las computadoras, preservando la identidad del carácter original en su nueva representación.

Es interesante que existen morfismos que no son fáciles de evaluar y hacen que un problema que es fácil, se transforme en uno difícil o viceversa. Por ejemplo, el problema del logaritmo discreto es fácil si la representación de los elementos es en $\mathbb{Z}_{n-1}$ o en $(\mathbb{Z}_n)^\times$

2.3. Congruencias

La teoría de congruencias es un pilar fundamental de la teoría de números y desempeña un papel crucial en criptografía y otras aplicaciones matemáticas. La relación de congruencia se define de la siguiente manera:

Sea $a, b, n \in \mathbb{Z}$ con $n > 0$. Decimos que a es congruente con b módulo n, y lo denotamos por:

$$a \equiv b \pmod{n},$$

si $n \mid (a - b)$, es decir, si $a - b$ es un múltiplo de n. Intuitivamente, esto significa que a y b tienen el mismo residuo cuando se dividen entre n.

2.3.1. Propiedades Básicas de las Congruencias

Las congruencias disfrutan de varias propiedades fundamentales que permiten operar con ellas de manera similar a las ecuaciones algebraicas habituales. Para $a, b, c \in \mathbb{Z}$ y $n > 0$:

- **Reflexividad:** $a \equiv a \pmod{n}$.
- **Simetría:** Si $a \equiv b \pmod{n}$, entonces $b \equiv a \pmod{n}$.
- **Transitividad:** Si $a \equiv b \pmod{n}$ y $b \equiv c \pmod{n}$, entonces $a \equiv c \pmod{n}$.

Además, las congruencias permiten realizar operaciones aritméticas:

- **Adición y sustracción:** Si $a \equiv b \pmod{n}$ y $c \equiv d \pmod{n}$, entonces:

$$a + c \equiv b + d \pmod{n}, \quad a - c \equiv b - d \pmod{n}.$$

- **Multiplicación:** Si $a \equiv b \pmod{n}$ y $c \equiv d \pmod{n}$, entonces:

$$a \cdot c \equiv b \cdot d \pmod{n}.$$

- **Potencias:** Si $a \equiv b \pmod{n}$, entonces para cualquier $k \geq 0$:

$$a^k \equiv b^k \pmod{n}.$$

2.3.2. Inversos Multiplicativos

En el criptosistema de Paillier, en el anillo de clases de restos $\mathbb{Z}/n$ que contiene los mensajes en texto claro, un número a tiene un inverso multiplicativo si existe un b tal que:

$$a \cdot b \equiv 1 \pmod{n}.$$

Tal inverso existe si y solo si a y n son coprimos, es decir, $\gcd(a, n) = 1$. La existencia de inversos es un concepto clave para definir el grupo multiplicativo $(\mathbb{Z}/n\mathbb{Z})^\times$, compuesto por los elementos de $\mathbb{Z}/n\mathbb{Z}$ que tienen inverso multiplicativo.

2.4. El pequeño teorema de Fermat

El Pequeño Teorema de Fermat es una herramienta esencial en la teoría de números y tiene aplicaciones importantes en criptografía, especialmente en algoritmos como RSA. El teorema establece lo siguiente:

Si p es un número primo y a es un entero tal que p no divide a a, entonces:

$$a^{p-1} \equiv 1 \pmod{p}.$$

En términos del grupo multiplicativo $(\mathbb{Z}_p)^\times$, el teorema garantiza que para cualquier $a \in (\mathbb{Z}_p)^\times$, el orden de a divide $p - 1$, que es el orden del grupo.

Este resultado tiene las siguientes consecuencias prácticas. Si a es cualquier entero, no necesariamente coprimo con p, entonces:

$$a^p \equiv a \pmod{p}.$$

Esta generalización simplifica cálculos modulares en diversas aplicaciones, incluyendo sistemas de autenticación y firmas digitales.

Demostración del pequeño teorema de Fermat. Consideremos el conjunto $\mathbb{Z}_p^\times = \{1, 2, \ldots, p-1\}$ con la operación de multiplicación módulo p. Este conjunto forma un grupo bajo la multiplicación, ya que:

- $\mathbb{Z}_p^\times$ es cerrado bajo la operación de multiplicación.
- Existe un elemento neutro, 1, tal que $a \cdot 1 \equiv a \pmod{p}$ para todo $a \in \mathbb{Z}_p^\times$.
- Cada elemento $a \in \mathbb{Z}_p^*$ tiene un inverso multiplicativo b tal que $a \cdot b \equiv 1 \pmod{p}$.
- La multiplicación es asociativa y conmutativa (es un grupo abeliano).

Sea a un elemento de $\mathbb{Z}_p^*$ (es decir, $1 \leq a \leq p - 1$). Definimos $k = |a|$, el orden de a, es decir, el menor entero positivo k tal que:

$$a^k \equiv 1 \pmod{p}.$$

Por el Teorema de Lagrange [2], el orden k de a debe dividir el orden del grupo $|\mathbb{Z}_p^*| = p - 1$. Por lo tanto, podemos escribir $p - 1 = k \cdot m$ para algún entero m. Entonces:

$$a^{p-1} = a^{k \cdot m} = (a^k)^m \equiv 1^m \equiv 1 \pmod{p}.$$

Esto demuestra que para cualquier $a \in \mathbb{Z}_p^*$, se cumple $a^{p-1} \equiv 1 \pmod{p}$, completando la demostración. □

2.5. Encriptación y Cifrados

Antes de empezar la explicación de los criptosistemas a implementar, vamos a explorar uno de los cifrados más sencillos y primitivos que existen para comenzar a tener una idea de como se emplean los conceptos de grupos y aplicaciones en la criptografía.

Ejemplo 3 (Cifrado César)**.** *La nomenclatura tiene origen histórico por el cual se le atribuye a Julio César haber usado este tipo de cifrado para enviar mensajes importantes de contenido militar.*

Considerando el alfabeto latino $\Sigma = \{A, ..., Z\}$ *de 26 letras, si se cifra un mensaje con el cifrado César con desplazamiento 3, entonces la A se sustituye por la D, la B por la E... y así sucesivamente, es decir, se sustituye por la tercera palabra en orden alfabético a partir de la letra considerada.*

Observemos que podemos identificar cada letra del alfabeto por los elementos del grupo $\mathbb{Z}_26$*, y el cifrado César de desplazamiento 3 sería equivalente a sumar 3 a cada número/letra del mensaje.*

Uno de los problemas de este tipo de criptosistemas es su falta de robustez, puesto que sólo pueden existir 26 cifrados distintos y aun prescindiendo de la ayuda de un ordenador, es fácil poder romper un cifrado César.

Una manera para evitar este tipo de situaciones es empleando una encriptación mediante problemas más complejos.

2.5.1. Encriptación Homomórfica

Los **cifrados homomórficos** permiten realizar operaciones aritméticas directamente sobre los textos cifrados, sin necesidad de descifrarlos previamente.

Es decir, dados dos mensajes $m_1, m_2 \in M$, donde M representa el espacio de los mensajes en claro (sin cifrar), y sus correspondientes textos cifrados $c_1, c_2 \in C$, donde C es el espacio de los textos cifrados, y tanto M como C son grupos, se cumple la siguiente propiedad:

$$\text{Desencriptación}(c_1 * c_2) = m_1 * m_2$$

También se puede definir como que tanto la encriptación como la desencriptación son morfismos. Esta propiedad garantiza que las operaciones realizadas en el espacio de los textos cifrados se reflejen correctamente en el espacio de los mensajes sin cifrar, manteniendo la privacidad de los datos.

Es un problema complicado tener un criptosistema seguro, eficiente y que ofrezca esta propiedad, por ello hay soluciones parciales que son usables. Dependiendo de la cantidad de operaciones que soporte un criptosistema homomórfico, se pueden distinguir las siguientes clases:

- **Partially Homomorphic Encryption (PHE)**. Estos criptosistemas soportan un número ilimitado de veces la operación de suma o multiplicación en los datos cifrados, pero no ambas. A este grupo pertenecen los criptosistemas de Paillier y Damgard-Jurik, que son aditivamente homomórficos. También hay criptosistemas multiplicativamente homomórficos como el de RSA, presentado por Rivest et al. [1978], o el de ElGamal [1985].
- **Somewhat Homomorphic Encryptioin (SHE)**. Soporta tanto operaciones de suma como multiplicación, pero con limitaciones sobre la profundidad y la cantidad de las operaciones.
- **Fully Homomorphic Encryption**. Permiten tanto la operación de suma como multiplicación un número ilimitado de veces sobre los datos cifrados. Hasta hace relativamente poco (2009) no se había demostrado si este esquema era posible, pero en su tesis Gentry [2009] demostró que esto era posible mediante el concepto matemático de *lattices*/retículos.

2.6. Criptografía de Clave Pública

Los protocolos de PSI emplean técnicas criptográficas avanzadas, como la criptografía de clave pública, la criptografía de curva elíptica y los esquemas de homomorfismo criptográfico, para asegurar

que la intersección de los conjuntos se realice de manera segura y privada. En este trabajo, se desarrollarán dos implementaciones de sistemas de clave pública homomórficos, específicamente los esquemas Paillier y Damgård-Jurik.

La criptografía de clave pública, también conocida como criptografía asimétrica debido a la existencia de un par de claves distintas por cada participante en el proceso comunicación, es un sistema criptográfico para la transmisión de información segura por un medio inseguro.

En contraposición, en la criptografía simétrica solo existe una *clave secreta* que deben poseer ambos participantes del proceso de comunicación. Debido a esto, la criptografía asimétrica es más segura debido a que cada entidad gestiona su propia clave secreta. Esto no significa que la criptografía simétrica no se use hoy en día. Por ejemplo, AES todavía es usado en la actualidad, ya que la generación sistemas de clave pública tiene un coste considerable.

La principal idea detrás de un esquema de criptografía de clave pública fue introducido por primera vez en Diffie y Hellman [1976]. En lugar de utilizar una clave compartida como en la criptografía simétrica, propusieron el uso de dos claves relacionadas, pero distintas: una clave pública y una clave privada. Estas claves están vinculadas por una función matemática de *trapdoor function* que es fácil de calcular en una dirección pero extremadamente difícil de invertir sin información adicional, como la clave privada.

Esto permitió el desarrollo del intercambio seguro de claves, donde dos partes pueden establecer una clave de sesión segura a través de un canal de comunicación inseguro sin compartir previamente ninguna información secreta. La seguridad de este enfoque se basa en la dificultad de resolver ciertos problemas matemáticos intratables.

2.6.1. Criptosistema RSA

Poco después de que Diffie y Hellman introdujeran la idea de criptografía de clave pública, Rivest et al. [1978] propusieron un esquema que permitía tanto cifrados como firmas digitales. Debido a su relevancia y a que cuenta con la propiedad de homomorfismo multiplicativo, me ha parecido interesante su análisis.

Propuesto por Rivest, Shamir y Adleman en 1978, el *RSA* es un sistema criptográfico basado en la factorización de números enteros grandes, ofreciendo homomorfismo multiplicativo.

Generación de claves

Para generar un par de claves RSA se siguen los siguientes pasos:

1. Seleccionar dos números primos grandes p y q.
2. Calcular $N = p \times q$

   y

   $\phi(N) = (p-1) \times (q-1)$.
3. Escoger un número e coprimo con $\phi(N)$.
4. Calcular el inverso modular d de e módulo $\phi(N)$.

Clave pública: (N, e).
Clave privada: (N, d).

Cifrado

Para cifrar un mensaje M en utilizando la clave pública (N, e) es necesario computar la siguiente operación para obtener el cifrado (C):

$$C = M^e \mod N$$

Descifrado

Para descifrar el mensaje cifrado C utilizando la clave privada (N, d), se realiza la siguiente operación:

$$M = C^d \mod N$$

Firma digital

RSA permite también firmar digitalmente un mensaje M utilizando la clave privada (N, d). Para verificar dicha firma se puede emplear la clave pública (N, e):

1. Firmar: $S = M^d \mod N$.
2. Verificar: $S^e \mod N$.

Homomorfismo multiplicativo

Cabe destacar, que RSA es un criptosistema multiplicativamente homomorfo, es decir, que si multiplicas los cifrados de dos textos, obtienes el cifrado del producto de ambos textos. No obstante, en este caso RSA es determinista, es decir, una encriptación sobre un mismo texto con la misma clave pública da el mismo texto cifrado en cada computación.

Para evitar esto, se emplean *padding schemes* añadiendo aleatoriedad al proceso. Sin embargo, esto rompe"las propiedades homomórficas del criptosistema. Por ello, es que para implementar el esquema PSI se usan los criptosistema de Paillier y Damgard-Jurik, que no son deterministas y tienen la propiedad de ser homomórficos en la adición.

2.6.2. Criptosistema de Paillier

El criptosistema de Paillier es un sistema de cifrado asimétrico basado en la dificultad de calcular las clases del n-residuo, considerado un problema computacionalmente difícil. Fue propuesto por Paillier [1999] y es conocido por su propiedad de homomorfismo aditivo, lo que significa que las operaciones en texto cifrado producen resultados equivalentes a las operaciones en texto claro.

En este criptosistema se maneja para los mensajes en claro en anillo de clases de restos Z_n y los textos cifrados están el grupo de unidades del anillo de clases de restos Z_{n^2}, donde n es el producto entre dos números primos grandes.

Generación de claves

A igual que en RSA, para generar las claves en el criptosistema de Paillier se seleccionan un par de números primos grandes p y q, pero con algunas diferencias:

1. Selecciona dos números primos grandes p y q.
2. Calcula el módulo $n = p \times q$.
3. Calcula $\lambda = \text{lcm}(p-1, q-1)$, donde lcm es el mínimo común múltiplo.
4. Elige un número aleatorio g tal que g sea coprimo con n^2.
5. La clave pública es (n, g) y la clave privada es (λ, μ), donde μ es el inverso multiplicativo de $L(g^\lambda \mod n^2)$ módulo n.

Cifrado

Para cifrar un mensaje m seguimos el siguiente procedimiento, pero usando el texto a cifrar como exponente en vez de como base:

1. Elige un número aleatorio r tal que $0 < r < n$ y r sea coprimo con n. Esta fase permites que cada cifrado de un mismo texto con la misma clave pública produzca resultados diferentes la gran mayoría de las veces.

2. Calcula el texto cifrado c como $c = g^m \cdot r^n \mod n^2$.

Descifrado

Para descifrar un texto cifrado c, se sigue el siguiente procedimiento:

1. Calcula $L(c^\lambda \mod n^2)$. La función L es $[L(x) = \frac{x-1}{n}]$.

2. Calcula el texto plano m como $m = L(c^\lambda \text{ mód } n^2) \cdot \mu \text{ mód } n$.

Homomorfismo aditivo

Una de las propiedades más importantes del criptosistema de Paillier es su homomorfismo aditivo. Esto significa que si se tienen dos textos cifrados c_1 y c_2 que cifran los mensajes m_1 y m_2 respectivamente, entonces el producto de c_1 y c_2 (módulo n^2) también cifra la suma de m_1 y m_2. Las operaciones soportadas son:

- Adición entre dos números cifrados:

$$c_add = c_1 \cdot c_2 \mod n^2$$

- Adición entre un número cifrado y un escalar:

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

  Siendo g parte de la clave pública (n, g).

- Multiplicación entre un número cifrado y un escalar:

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

2.6.3. Criptosistema de Damgard-Jurik

El criptosistema de Damgard-Jurik es una generalización del criptosistema de Paillier. Realiza sus computaciones en el módulo n^{s+1}, donde $n = p \times q$ y s es un parámetro que define el número de niveles de seguridad del sistema. El caso en el que $s = 1$ corresponde al criptosistema de Paillier, mientras que valores mayores de s proporcionan mayor seguridad.

Al igual que en Paillier, este criptosistema maneja los mensajes en claro en el anillo de clases de restos Z_n, pero a la diferencia está en los textos cifrados que se encuentran en el grupo de unidades del anillo de clases de restos $Z_{n^{s+1}}$. Gracias a esto, se puede definir el factor de expansión de los textos cifrados mediante la elección de s

Generación de claves

Para generar las claves en el criptosistema de Damgard-Jurik, se siguen los siguientes pasos:

1. Selecciona dos números primos grandes p y q.
2. Calcula el módulo $n = p \times q$.
3. Calcula n^{s+1}, que será el módulo para las operaciones.
4. Calcula $\lambda = \text{lcm}(p-1, q-1)$, donde lcm es el mínimo común múltiplo.
5. Elige un número aleatorio g tal que g sea coprimo con n^{s+1}. Adicionalmente, se debe cumplir que $g = (1+n)^j \cdot x \mod n^{s+1}$ para dos parámetros j y x, que pueden ser seleccionados aleatoriamente.
6. La clave pública es (n^{s+1}, g) y la clave privada es (λ, μ), donde μ es el inverso multiplicativo de $L(g^\lambda \mod n^{s+1})$ módulo n, y $L(x) = \frac{x-1}{n}$ es la función L utilizada en la operación de descifrado.

Cifrado

Para cifrar un mensaje m, el procedimiento es el siguiente:

1. Elige un número aleatorio r tal que $0 < r < n^{s+1}$ y r sea coprimo con n^{s+1}.
2. Calcula el texto cifrado c como:
$$c = g^m \cdot r^{n^s} \mod n^{s+1}$$
Este cifrado garantiza que, incluso cuando el mensaje m sea el mismo, los resultados cifrados serán diferentes debido al número aleatorio r.

Descifrado

Para descifrar un texto cifrado c, se sigue el siguiente procedimiento:

1. Calcula $L(c^\lambda \mod n^{s+1})$, donde $L(x) = \frac{x-1}{n}$.
2. Calcula el texto plano m como:
$$m = L(c^\lambda \mod n^{s+1}) \cdot \mu \mod n$$

Homomorfismo aditivo

El criptosistema de Damgard-Jurik conserva la propiedad de homomorfismo aditivo, al igual que el criptosistema de Paillier. Esto significa que, dados dos textos cifrados c_1 y c_2 que cifran los mensajes m_1 y m_2 respectivamente, el producto $c_1 \cdot c_2 \mod n^{s+1}$ cifra la suma de los mensajes $m_1 + m_2$.
Las operaciones soportadas son:

- **Adición entre dos números cifrados:**
$$c_{\text{add}} = c_1 \cdot c_2 \mod n^{s+1}$$
- **Adición entre un número cifrado y un escalar:**
$$c_{\text{add}}^k = c \cdot (g^k \mod n^{s+1}) \mod n^{s+1}$$
donde g es parte de la clave pública (n^{s+1}, g).
- **Multiplicación entre un número cifrado y un escalar:**
$$c_{\text{mul}} = c^k \mod n^{s+1}$$

Simplificación del Criptosistema de Damgård-Jurik

El criptosistema de Damgård-Jurik puede ser simplificado de manera que ya no incluya al criptosistema de Paillier como un caso especial. A continuación, se presenta el proceso de simplificación:

- La base g se fija como $g = n + 1$.
- El exponente de descifrado d se calcula tal que $d = 1 \mod n^s$ y $d = 0 \mod \lambda$.

En este caso, el descifrado produce

$$c^d = (1 + n)^m \mod n^{s+1}.$$

Empleando el algoritmo de desencriptación de Paillier de forma recursiva, obtenemos el texto plano m.

3 Private Set Intersection

La Intersección Privada de Conjuntos (PSI) es un protocolo criptográfico diseñado para permitir que dos entidades, cada una con un conjunto de datos privado, puedan determinar la intersección de sus conjuntos sin revelar ninguna información adicional más allá de los elementos comunes.

Empresas tecnológicas de renombre, como Apple y Google, ya han implementado técnicas avanzadas relacionadas con la intersección privada de conjuntos (PSI) para proteger la privacidad en sus servicios. Estas aplicaciones no solo garantizan seguridad, sino que también abren nuevas posibilidades en la gestión eficiente de datos confidenciales.

Un ejemplo relevante es el documento técnico de Apple, donde se describe el uso de PSI en el contexto de la detección de Material de Abuso Sexual Infantil (CSAM, por sus siglas en inglés). El enfoque utiliza algoritmos criptográficos para identificar coincidencias con bases de datos de huellas digitales sin comprometer la privacidad del usuario Inc. [2021].

Google, en su extensión de *Password Checkup*, emplea técnicas basadas en PSI para permitir a los usuarios verificar si sus credenciales han sido comprometidas, sin exponer ni las contraseñas del usuario ni las bases de datos utilizadas para la comparación Google [2021].

3.1. Esquemas de PSI

Como ya se comentó anteriormente, hay diversas maneras, técnicas, esquemas, protocolos y primitivas que se pueden usar para implementar PSI. Estos métodos permiten que dos partes colaboren en la comparación de sus conjuntos sin revelar información adicional sobre los elementos no comunes. A continuación, se describen los tres enfoques PSI que se van a implementar y estudiar.

3.1.1. Oblivious Polynomial Evaluation PSI

Basado en *Efficient private matching and set intersection* (Freedman et al. [2004]), este esquema utiliza la evaluación polinómica inconsciente (Oblivious Polynomial Evaluation) para realizar la intersección privada de conjuntos. El protocolo se basa en que ambas partes calculan y evalúan un polinomio de manera conjunta sin revelar ningún detalle sobre los elementos individuales de los conjuntos, salvo aquellos que coinciden.

En este trabajo, vamos a emplear la criptografía homomórfica, mediante el uso de los criptosistemas de Paillier y Damgard-Jurik, para garantizar la privacidad de los conjuntos involucrados en la computación.

Dado dos conjuntos de datos privados, $C = \{c_1, \ldots, c_n\}$ y $S = \{b_1, \ldots, b_m\}$, el proceso para computar la intersección de conjuntos usando criptografía homomórfica es el siguiente::

1. En primer lugar, C calcula el polinomio $p_C(X)$ cuyas raíces son los elementos de su conjunto de datos:

$$p_C(X) = \prod_{i=1}^{n}(X - c_i) = \sum_{i=0}^{n} p_i X^i, \quad \forall i = 0, \ldots, n.$$

A continuación, C encripta cada coeficiente p_i del polinomio, obteniendo $E(p_i)$, y envía tanto los coeficientes encriptados como la clave pública utilizada para la encriptación a S.

2. Una vez que S recibe el polinomio encriptado, utiliza las propiedades de la criptografía homomórfica junto con el algoritmo de Horner para evaluar el polinomio en cada uno de sus elementos b_i:

$$\forall b_i \in S, \quad E_i(p_C(b_i)).$$

De esta forma, S evalúa el polinomio en sus propios elementos, manteniendo la privacidad de los datos de C y S gracias a la criptografía homomórfica.

3. Si un elemento de S se encuentra en C, el resultado de la evaluación del polinomio será cero (encriptado). Para poder computar la intersección, al resultado de la evaluación del polinomio con el elemento b_i se le suma el mismo elemento b_i, pero encriptado, aprovechando las propiedades de la criptografía homomórfica. Tras esto, S envía los resultados de las evaluaciones encriptadas de vuelta a C.

4. Al recibir las evaluaciones encriptadas de S, lo único que tiene que hacer C es desencriptarlas utilizando su clave privada y comprobar si el elemento está presente en su conjunto. La operación de desencriptación se realiza de la siguiente manera:

$$\forall i \in \{1, \ldots, n\}, \quad D(p_i) = E_i$$

donde D es la función de desencriptación de C. Finalmente, C comprueba si $b_i \in C$, es decir, si el valor desencriptado corresponde a uno de los elementos de su conjunto.

A lo largo del protocolo ni S ni C obtienen información adicional de su contraparte. C recibe solo las evaluaciones de S y solo obtiene los elementos de la intersección. En el caso de S, este no sabe en ningún momento si un elemento de su conjunto es raíz de un polinomio al evaluarlo, al estar los resultados encriptados.

No obstante, hay que tener en cuenta que, como en la mayoría de los protocolos PSI, siempre cabe la posibilidad de que un usuario malicioso defina su conjunto con todos los valores posibles. En este caso se obtendría el conjunto del adversario. Para evitar esto se puede limitar el grado máximo del polinomio.

Es importante recordar que en el caso expuesto S tiene que usar siempre la clave pública de C para realizar las operaciones homomórficas durante el proces, ya que la desencriptación del resultado es mediante la clave privada de C.

Finalmente, se han comentado las extensiones posibles de PSI, como PSI-CA, en el que lo que interesa solo es la cardinalidad de la intersección y no el contenido. En este protocolo dicha implementación es trivial, ya que en el paso tres S solo tiene que no sumar nada a la evaluación y devolverá cero si es raíz. De esta manera se puede saber la cardinalidad de la intersección, pero no conocer los elementos en sí.

3.1.2. Bit Vector PSI

La siguiente implementación sigue el modelo presentado en *New approach to set representation and practical private set-intersection protocols* (Ruan et al. [2019]). Esta implementación está dirigida sobre todo para servidores en la nube con un conjunto de datos considerable.

El artículo presenta un nuevo enfoque para representar conjuntos mediante vectores de bits. Un conjunto X seleccionado de un dominio fijo con n elementos $(m_1, \ldots, m_n)$ se denota como un vector de bits $(b_1, \ldots, b_n)$, donde si el bit $b_i = 1$ $(1 \leq i \leq n)$ significa que $s_i \in X$, de lo contrario $s_i \notin X$. Con esta representación, podemos obtener la intersección de conjuntos y la cardinalidad de la intersección mediante operaciones sobre los vectores. Este enfoque es particularmente adecuado para entornos de

computación en la nube y oculta naturalmente la cardinalidad de un conjunto, ya que la longitud del vector de bits siempre es n.

El funcionamiento del esquema es el siguiente:

1. Existe una precomputación antes de iniciar el protocolo. Las entidades S y C representan sus conjuntos $\{S\}$ y $\{C\}$ como vectores de bits b_S y b_C, respectivamente. Ambas partes generan sus pares de claves públicas/privadas (pk_S, sk_S) y (pk_C, sk_C). Luego, cada entidad cifra cada bit b_{Si} y b_{Ci} de sus vectores de bits con sus respectivas claves públicas:

$$c_{Si} = \mathrm{En}^{pk_S}(b_{Si}), \quad c_{Ci} = \mathrm{En}^{pk_C}(b_{Ci})$$

   Estos cifrados se almacenan, para no tener que volver a computarlos al realizar PSI entre 2 organizaciones.

2. Para comenzar el proceso de PSI, C debe enviar su clave pública y su conjunto de datos, representado como un *bit vector* encriptado ($c_C = \mathrm{En}^{pk_C}(b_C)$), a el servidor S.

3. Al recibir (c_C), S computa los valores mediante:

$$\forall b_{Si} \in S \quad e_i = (c_{Ci})^{b_{Si}} \cdot \mathrm{En}^{pk_C}(0)$$

   y envía los resultados (e_i) de vuelta a S. La computación se basa en aplicar la multiplicación entre un número cifrado y un escalar, que es una operación soportada por los criptosistemas de Paillier y Damgard-Jurik. La adición de un cero encriptado tiene como finalidad preservar apropiadamente la privacidad de S, asegurando que no se revele el valor de la evaluación en ningún momento, incluso si el valor de b_{Si} es 0.

4. Finalmente, C descifra los valores e_i usando su clave privada sk_C, obteniendo:

$$b_{Ci} = \mathrm{Dec}^{sk_C}(e_i)$$

   Entonces, si $b_i = 1$, significa que $m_i \in \{S\} \cap \{C\}$.

   En caso contrario, si $b_i = 0$, significa que $m_i \notin \{S\} \cap \{C\}$.

La principal ventaja que introduce este esquema es la reducción significativa de cálculos complejos, ya que todos los cifrados pueden ser precomputados, lo que lo hace más eficiente que otros protocolos PSI. Además, los datos se mantienen seguros, ya que se utilizan directamente los conjuntos cifrados para realizar el cálculo de la intersección.

Al igual que en el protocolo de Oblivious Polynomial Evaluation PSI, esta implementación puede ser extendida para desvelar solo la cardinalidad de la intersección. Al computar la intersección mediante la multiplicación de ambos *bit vectors*, es posible sumar todos los elementos de dicha intersección y devolver así solo la cardinalidad de la intersección. Para ello, en el paso 3, Al recibir S el vector de bits encriptado c_{Ci} del cliente C, se tiene que realizar la siguiente computación:

$$e = \prod_{i=1}^{n} \left(c_{Ci}^{b_{Si}}\right) \cdot \mathrm{En}^{pk_c}(0),$$

y devolver dicho resultado (e) al usuario C.

3.1.3. Bloom Filter PSI

El siguiente protocolo está basado en el artículo *A privacy-preserving fuzzy interest matching protocol for friends finding in social networks*. En este los autores proponen un protocolo PSI para encontrar amigos en redes sociales. Para ello se hace uso de *Bloom Filters*, una estructura de datos eficiente en términos de espacio que permite representar un conjunto de elementos y verificar si un elemento pertenece a dicho conjunto con una pequeña probabilidad de error debido a que puede haber falsos positivos, aunque la existencia de falsos negativos es imposible.

El protocolo es similar al basado en vectores de bits; sin embargo, ofrece una mayor seguridad gracias a la incorporación de *Bloom Filters*. Esta mejora facilita también la extensión del protocolo para permitir que la computación sea delegada a un tercero, manteniendo la privacidad de los datos.

Introducción Bloom Filter

Un *Bloom Filter* está compuesto por:

- Un conjunto de elementos $X = \{x_1, x_2, \ldots, x_n\}$ perteneciente al dominio.
- Un vector de bits $B = (b_1, b_2, \ldots, b_m)$ de tamaño m, inicializado a 0.
- Un conjunto de k funciones hash distintas $(h_1, h_2, \ldots, h_k)$.

Para representar el conjunto, es necesario aplicar sobre cada elemento $x_i \in X$, las funciones hash existentes, generando cada función un índice en el rango $[1, m]$. El contenido en dichas posiciones del vector se ponen a 1.

Un elemento se encuentra en la lista si tras aplicar sobre el todas las funciones hash todos los elementos del vector tienen un 1.

Para verificar si un elemento y pertenece al conjunto X, se calcula $h_j(y)$ para cada j y se verifica si los bits correspondientes en B son todos 1. Si al menos uno de los bits es 0, entonces $y \notin X$. Sin embargo, incluso si todos los bits son 1, existe una pequeña probabilidad de que y no esté realmente en X, lo que se denomina un falso positivo.

Protocolo PSI basado en Bloom Filters.

El esquema PSI opera de la siguiente manera:

1. Existe una precomputación en las que los usuarios C y S generan sus *Bloom Filters* BF_C y BG_S, respectivamente. Tras esto se cifran todos los elementos de los*Bloom Filters* empleando un criptosistema homomórfico en $E(BF_C)$ y $E(BF_S)$.

2. Para iniciar el protocolo PSI, una de las entidades, por ejemplo C, envía su *Bloom Filter* cifrado $E(BF_C)$ al otro usuario S,

3. El servidor S al recibir $(E(BF_C)$ aplica la suma entre elemento cifrado y escalar a cada elemento i del vector cifrado del cliente con su opuesto del conjunto del servidor. Si dos elementos del *Bloom Filter* suman 2 hay intersección. Para poder distinguir si hay o no intersección, le "restamos" 2 a la suma obtenida:

$$\begin{aligned} & E(B_{FC}) \cdot g^{B_{FS}} \cdot E(-2)^r \\ = {} & E\big[r_0(B_{FC}[0] + B_{FS}[0] - 2, \ldots, \\ & r_{m-1}[B_{FC}[m-1] + B_{FS}[m-1] - 2]\big)] \\ = {} & E(r(B_{FC \cup S}/2) \end{aligned}$$

Una vez computado $E(r(B_{FC\cup S}/2)$ se envía el resultado computado a C.

Observación 4. *Durante la implementación de esta computación se observó que en esta paso era posible eliminar la necesidad de realizar una "resta" de 2 unidades, si en vez de realizar la suma entre un elemento cifrado y un escalarse multiplicase el elemento encriptado por el escalar (B_FS) directamente. Se probó y el sistema funcionaba de manera correcta. No obstante, para seguir el protocolo establecido se ha decidido seguir el modelo definido por el artículo.*

4. El cliente C, al recibir el resultado encriptado, solo tiene que desencriptar cada elemento del *Bloom Filter*. Si el valor desencriptado de $r_0([B_{FC}[0] + B_{FS}[0] - 2) = 0$ significa que $BF_C[i] = BF_S[i] = 1$ en caso contario no hay intersección y se pone un 0 en dicha posición del *Bloom Filter*.

En el artículo presentan eficiencia de espacio como una de las ventajas de los *Bloom Filters*, diciendo que escala linealmente con el tamaño del mismo. No obstante, el tamaño del vector resultado es más grande que en el caso de la implementación de vectores de bits para un mismo dominio, debido a que el *Bloom Filter* debe tener un tamaño superior al del dominio para evitar falsos positivos. Además, si se quiere una probabilidad de falsos positivos despreciable, el tamaño del *Bloom Filter aumenta considerablemente*.

En general, este protocolo presentado cuenta con diversas limitaciones como la posibilidad de falsos positivos debido al *Bloom Filter* así como la necesidad de más operaciones homomórficas, pero cuenta con la ventaja de que en la fase de evaluación no es necesario que el servidor encripte su propio conjunto de datos.

3.2. Variantes de PSI

En esta sección se exponen implementaciones alternativas de PSI empleando otras primitivas y protocolos que no se han analizado ni estudiado debido a que estaban fuera del alcance definido del proyecto.

Algunas de las extensiones y variantes más interesantes son las siguientes:

- **Multi-party PSI**. Se basa en aplicar PSI a más de 2 entidades. Algunas de las implementaciones presentadas se basan en aplicar topologías de red entre los participantes del protocolo, como una distribución en anillo o en estrella.
- **O-PSI**. Es una variante cuya principal prioridad es diseñar un protocolo lo bastante seguro para que sea posible delegar la computación del PSI a una entidad externa, preservando de todas maneras la privacidad de los participantes del PSI.

4 Implementación y Evaluación

En esta sección se explica el proceso seguido para el desarrollo de la aplicación, así como la metodología seguida para realizar las pruebas y un análisis de los resultados obtenidos.

4.1. Análisis de Requisitos

Se ha estudiado y analizado los objetivos y funcionalidades de la herramienta a implementar. A partir de este análisis se han presentado los siguientes requisitos. Hay que comentar, que algunos de estos requisitos han sido modificados a lo largo del desarrollo del proyecto, eliminando y añadiendo nuevas funcionalidades en función de los problemas o necesidades que han ido apareciendo.

En el cuadro 1 se pueden observar los requisitos funcionales del sistema que han sido definidos a lo largo del desarrollo del proyecto.

En el cuadro 2 se pueden observar y analizar los requisitos no funcionales que se han definido para la aplicación.

4.2. Herramientas

Para el desarrollo de proyecto, se estudiaron diversas alternativas para implementar la lógica a nivel de aplicación, siendo las más interesantes las siguientes:

- Python usando la librería Cryptography. Esta opción es interesante debido a que Python es un lenguaje con una sintaxis limpia y legible, además de que la librería presentada aporta operaciones criptográficas básicas.
- Python usando Sagemath. Esta opción es también porque admite multitud de métodos interesantes para la implementación de sistemas criptográficos.
- C++, empleando la librería Crypto++. Esta es la opción más profesional y avanzada. Para implementaciones de criptosistemas que buscan tener el máximo nivel de control y eficiencia se suele emplear C++, o incluso C.

Tras analizar las herramientas, se acabó eligiendo usar Python como lenguaje de programación empleando Sagemath. Se ha preferido esta opción a las otras debido a que Sage aporta métodos interesantes para la generación de números primos, así como diversas funciones criptográficas. Además, el uso de Python nos ayuda a cumplir los requisitos no funcionales relacionados con desarrollar los criptosistemas con Python (**RNF03**) y el uso *Flask* (**RNF05**) para la interfaz gráfica. El uso de *Sage* nos ayuda con las funciones que aporta a cumplir con el **RNF04**.

Se ha descartado Cryptography, ya que es una librería con funciones de más alto nivel, no tan interesante para mi caso de uso. Finalmente, se descartó C++ con Crypto++ debido a su complejidad.

Previo al desarrollo del sistema, se implementó en Sagemath una versión funcional, pero no segura, de una intersección de conjuntos empleando el criptosistema de Paillier mediante la plataforma de Cocalc.

En la figura 1 se puede observar como se implementan diversos métodos del criptosistema de Paillier siguiendo los pasas descritos en la sección 2.6.2. El método *generate_paillier_kyepair* sigue los

Código	Descripción del Requisito
RF01	Es necesario desarrollar una interfaz gráfica de usuario en la cual se puedan configurar los parámetros del *Private Set Intersection*.
RF02	El sistema debe permitir al usuario especificar manualmente un conjunto de datos tanto para servidor como para cliente.
RF03	El sistema debe permitir generar conjuntos de datos aleatorios, especificando un tamaño máximo de los conjuntos de datos, así como el valor máximo de cada elemento.
RF04	En la interfaz gráfica de usuario se podrán escoger entre las distintas implementaciones de *Private Set Intersection para computar la intersección*
RF05	También es necesario que se pueda especificar el sistema criptográfico homomórfico a emplear, así como parámetros del mismo como el tamaño de clave o el valor de*s* en el caso de Paillier.
RF06	Se pide que se muestre el resultado de la intersección en la interfaz gráfica de usuario.
RF07	Para el análisis de los criptosistemas hay que recopilar los datos temporales del PSI y mostrarlos en una tabla aparte.
RF08	Con la finalidad de facilitar el análisis de los datos obtenidos, hay que introducir la opción de exportar los datos obtenidos del *PSI* en formato *csv*.

Cuadro 1: Requisitos funcionales del sistema

Código	Descripción del Requisito
RNF01	Es necesario que la aplicación permita introducir nuevos sistemas criptográficos homomórficos para que puedan ser acoplados a los protocolos del PSI implementados.
RNF02	Es necesario que la interfaz gráfica de usuario sea intuitiva y fácil de usar, añadiendo una portada con documentación e instrucciones si fuese necesario.
RNF03	Usar el lenguaje Python para el desarrollo de los criptosistemas.
RNF04	Los criptosistemas tienen que ser implementados desde cero usando un lenguaje de aplicación. No se busca emplear librerías con los criptosistemas ya implementados como LightPHE.
RNF05	Emplear el *framework de Flask* para desarrollar la interfaz gráfica de usuario.

Cuadro 2: Requisitos no funcionales del sistema

pasos descritos, aunque para la elección de g se simplifica asignándole el valor de $n+1$, que es coprimo de n^2. Además, se almacenan valores de uso reiterado como *mu* y *lambda* para evitar computarlos de nuevo. La encriptación y desencriptación han sido implementadas siguiendo el esquema definido y se ha comprobado que funciona correctamente.

En cuanto al método *psi_paillier()* a pesar de que computa la intersección entre dos conjuntos, no preserva la privacidad de los mismos. Esto es claro porque en la implementación presentada *Alice* precisa de la lista desencriptada de *Bob* para la computación. Además, las claves públicas y privadas se comparten, algo que nunca debería de pasar. No obstante, el código implementado sigue siendo de gran ayuda para ver las capacidades de la criptografía homomórfica. La verificación de intersección se realiza mediante la ŕestaĺhomomórfica. A pesar de que Paillier solo soporte la adición en *ciphertexts*, si se suma el inverso modular de un número encriptado es como si se restase dicho número.

4.3. Desarrollo del Sistema

Previo al desarrollo de sistema, se mantuvieron reuniones con el tutor donde se realizó un análisis de los requisitos para especificar las características y prestaciones que debe tener la aplicación. En consecuencia, se han implementado los criptosistemas de Paillier o Damgard-Jurik para mantener la privacidad de los datos en el PSI.

Se han implementado también los 3 protocolos diferentes de PSI para analizar su eficiencia y desempeño en función de diversas métricas.

Además, los protocolos PSI implementados permitan devolver solo la cardinalidad de la intersección al computar el PSI si es necesario.

Para cumplir con el **RF01** y simplificar las pruebas, se ha desarrollado una aplicación web. En la imagen 2 se puede ver la página principal de la interfaz gráfica de la aplicación, con una pequeña descripción exponiendo sus funcionalidades, cumpliendo con el **RNF02**.

Esta aplicación permite definir los conjuntos del cliente y el servidor (**RF03**), además de poder generar conjuntos aleatorios para realizar las pruebas (**RF03**).

Además, dicha interfaz debe contar con opciones para poder seleccionar el protocolo PSI a usar, así como el tamaño de la clave y el criptosistema homomórfico a emplear como primitiva para la computación del PSI entre dos entidades (**RF04** Y **RF05**).

Finalmente, en la interfaz, los resultados de la intersección se muestran una vez computados para comprobar que el PSI se haya ejecutado correctamente (**RF06)**. Añadido a esto, la interfaz almacena los datos sobre la cantidad de tiempo que ha llevado cada paso del PSI, almacenándolos en una tabla **(RF07)** y permitiendo que estos sean exportados en formato *csv* **RF08** para facilitar su posterior análisis y estudio.

4.3.1. Estructura del Código

En la Ilustración 3 se puede observar un diagrama simplificado con el código desarrollado para realizar el estudio de las diferentes implementaciones de PSI. Se han obviado algunos elementos, como la definición de funciones criptográficas comunes y estructuras de datos propias, como la implementación de la clase BloomFilter, necesaria para la implementación de Private Set Intersection basado en la misma.

En general, se ha tomado la decisión de dividir el programa en distintos paquetes para facilitar el desarrollo del código y la modularidad del mismo cumpliendo con el requisito.

Con esta estructura se busca que sistema sea escalable y permita añadir más implementaciones de PSI o esquemas criptográficos de clave pública si es necesario, permitiendo implementaciones externas de criptosistemas y protocolos PSI al modelo mientras se ajusten a la estructura definida (**RNF01**).

- ***Schemes***. Este directorio contiene código relacionado con los esquemas y criptosistemas de clave pública que se van implementando. Se ha tomado la decisión de diseño de separar la *PublicKey* y la *PrivateKey* debido a que cada una de ellas tienen casos de uso diferentes. Además, de esta

```
from sage.all import *

def generate_paillier_keypair(n_length=32):
    p = random_prime(2^(n_length//2 - 1))
    q = random_prime(2^(n_length//2 - 1))
    n = p * q
    g = n + 1
    lambda_ = lcm(p-1, q-1)
    mu = inverse_mod(lambda_, n)
    public_key = (n, g)
    private_key = (lambda_, mu)
    return public_key, private_key

def paillier_encrypt(m, public_key):
    n, g = public_key
    r = randint(1, n-1)
    c = (power_mod(g, m, n^2) * power_mod(r, n, n^2)) % (n^2)
    return c

def paillier_decrypt(c, public_key, private_key):
    n, g = public_key
    lambda_, mu = private_key
    l = (power_mod(c, lambda_, n^2) - 1) // n
    m = (l * mu) % n
    return m

def psi_paillier(alice_set, bob_set, public_key, private_key):
    encrypted_alice_set = [paillier_encrypt(x, public_key) for x in alice_set]

    n, g = public_key

    enc_intersection = []
    for x in bob_set:
        # Computa el inverso modular del elemento para 'restar'
        enc_b = inverse_mod(paillier_encrypt(x, public_key), n)
        enc_sum = enc_b
        for enc_a in encrypted_alice_set:
            # Si esto es igual a cero significa que hay intersección
            enc_sum = enc_a * enc_b % n^2

            if paillier_decrypt(enc_suma, public_key, private_key) == 0
                enc_intersection.append(enc_b)

    decrypted_intersection = [paillier_decrypt(c, public_key, private_key) for c in
    intersection]

    intersection = set()
    for i, data in enumerate(decrypted_intersection):
        if i == 0:
            intersection.add(bob_set[i])

    return intersection

alice_set = [1, 2, 3, 4, 5]
bob_set = [3, 4, 5, 6, 7]

public_key, private_key = generate_paillier_keypair()

intersection = psi_paillier(alice_set, bob_set, public_key, private_key)
print("Intersection:", intersection)
```

Ilustración 1: Primer esbozo de la aplicación PSI en CoCalc

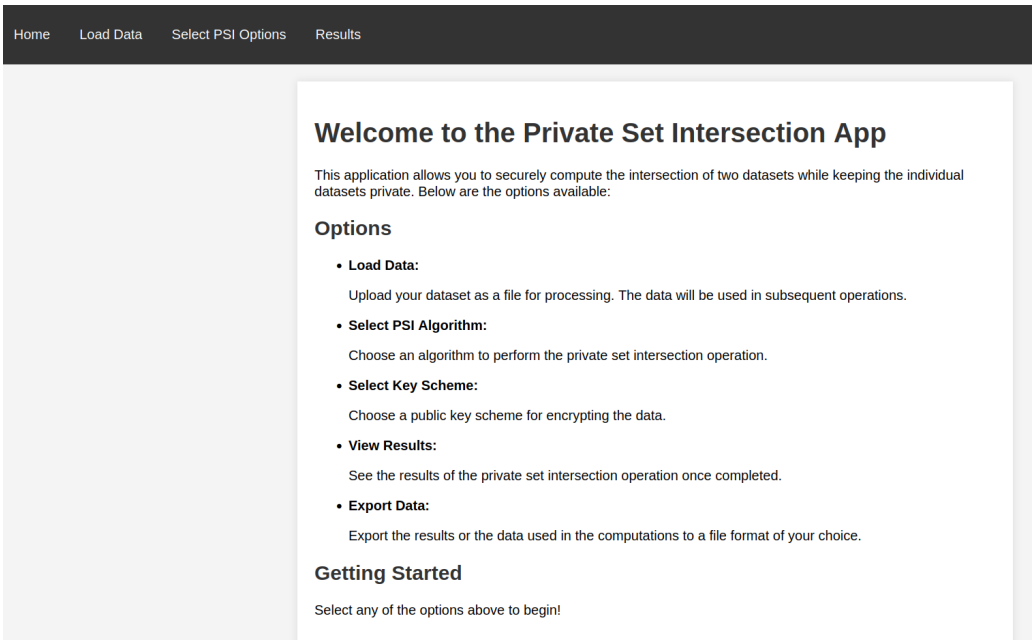

Ilustración 2: La página principal de la aplicación web con una pequeña guía de uso

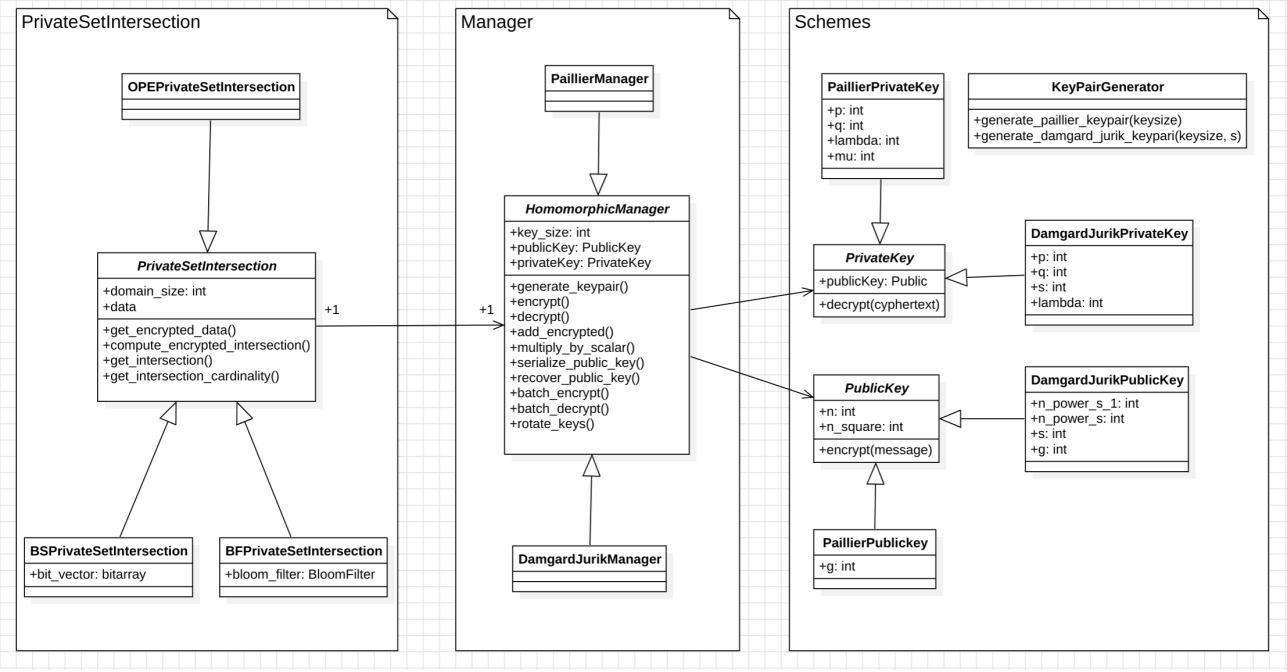

Ilustración 3: Diagrama con la estructura del código criptográfico

manera la *PrivateKey* se puede almacenarse por separada y aporta un nivel más de seguridad a la implementación.

Si se quiere añadir un nuevo criptosistema al paquete solo sería necesario definir una nueva clase *ElGamalPublicKey* y *ElGamalPrivateKey* que hereden de las clases base y definir un método en la clase estática *KeyPairGenerator* que se encarge de crear el par de claves cuando sea necesario.

- ***Manager***. Este paquete contiene el código asociado para gestionar y controlar el ciclo de vida de un par de claves públicas y privadas. En este se definen métodos criptográficos generales, así como herramientas para gestionar el ciclo de vida de las claves, permitiendo guardarlas en archivos, así como serializarlas.

  Se ha decidido este diseño, ya que permite para sistemas criptográficos distintos implementar métodos y prestaciones específicas. Además, de esta manera se consigue mantener el código relacionado con la *PublicKey* y *PrivateKey* estructurado y libre de funcionalidades innecesarias.

- ***PrivateSetIntersection***. Finalmente, tenemos la carpeta *PrivateSetIntersection* en la cual está almacenado el código dirigio a implementar distintos protocolos de Private Set Intersection. Esto permite modularidad en el código y para una implementación de un protocolo en particular, emplear diferentes sistemas criptográficos homomórficos para su gestión y desarrollo.

  Gracias a esta estructura, los nuevos protocolos de PSI definidos pueden heredar de la clase *PrivateSetIntersection* y añadir las características y estructuras de datos necesarias para su construcción en particular. El mejor ejemplo es el caso de *BFPrivateSetIntersection* en el que es necesario definir un *Bloom Filter* para su implementación.

Se han obviado partes del código en el esquema mostrado debido a que no son pertinentes para el análisis de las métricas de los protocolos PSI y ls sistemas criptográficos homomórficos.

Finalmente, cabe destacar que se ha desarrollado una sencilla interfaz gráfica mediante el *framework* de Flask. Para las pruebas y la toma de datos de los protocolos de PSI implementados se ha hecho uso de esta aplicación desarrollada. Esto ha facilitado la recopilación de los datos, ya que además permite exportar los datos almacenados en formato *csv*.

4.4. Pruebas Realizadas y Evaluación de Resultados

El objetivo de este trabajo es analizar y comparar el rendimiento de diferentes protocolos de Privates Set Intersection basados en criptografía homomórfica. Para ello se han implementado los criptosistemas de Paillier y Damgard Jurik así como la Intersección Privada de Conjuntos basada en *Oblivious Polynomial Evaluation*, vectores de bits y *Bloom Filters*.

Para realizar las pruebas se ha empleado en un Asus TUF Gaming F15 FX506LH-BQ116, con un procesador Intel Core i7-10870H (8 núcleos, 16 hilos, frecuencia base de 2.2 GHz y turbo de hasta 5.0 GHz, con 16 MB de caché). También cuenta con 16 GB de memoria RAM DDR4 a 2933 MHz.

A la hora de desarrollar las pruebas se ha intentado mantener el portátil con la menor cantidad de interferencias posibles y conectado a una fuente de alimentación estable. Además, debido a que solo nos interesan los tiempos de ejecución, se ha decidido obviar el intercambio de mensajes entre las entidades y se va a medir solo el tiempo de cómputo. Las pruebas que se han realizado se pueden dividir en dos apartados:

- En primer lugar, tenemos las pruebas relacionadas con el análisis temporal de los **criptosistemas** homomórficos. En estas se ha variado el tamaño de la clave (512, 1024, 2048, 3072, 4096, 8192) y se ha analizado los tiempos de ejecución de:

  - **Generación de claves**. Se ha medido el tiempo de cada criptosistema para generar un par de claves pública y privada.

- **Encriptación**. Se han tomado medidas del tiempo de encriptación de un valor numérico con una clave pública.
- **Desencriptación**. Para un cifrado se ha medido la cantidad de tiempo que se tarda en desencriptar con la clave privada.
- **Adición homomórfica**. Se ha medido el tiempo necesario para sumar dos números encriptados.
- **Adición cifrado con escalar**. Se ha medido el timepo empleado para sumar un cifrado con un escalar sin encriptar.
- **Multiplicación cifrado con escalar**. Se ha medido la cantidad de tiempo que tarda en computarse la multiplicación entre número encriptado escalar.

Gracias a que Damgard-Jurik es un criptosistema similar al de Paillier, por no decir que es una generalización del criptosistema de Paillier, las operaciones homomórficas soportadas son las mismas.

Para tomar los distintos tiempos de cada apartado, se ha generado un *script* que se encarga de medir los tiempos de creación de claves, así como de realizar las operaciones homomórficas necesarias. Este script permite especificar el número de iteraciones de este proceso. Tras esto se hace una media con los valores obtenidos y se construyen las gráficas finales.

- El siguiente bloque de pruebas se llevó a cabo utilizando la interfaz gráfica implementada con *Flask*. Estas pruebas se centraron en los datos temporales relacionados con la Intersección Privada de Conjuntos. En los tres protocolos implementados, es posible identificar tres etapas claramente diferenciadas:

  - **Fase 1 (Encriptación)**: El cliente necesita encriptar su conjunto de datos y enviarlo. En la implementación basada en vectores de bits y en *Bloom Filters* esta fase se precomputa.
  - **Fase 2 (Evaluación de los conjuntos)**: En esta fase el servidor, tras recibir el conjunto de datos, procede a evaluarlo mediante operaciones homomórficas según las pautas que establezca el protocolo. Una vez se ha evaluado el conjunto, se envía dicha evaluación de vuelta al cliente.
  - **Fase 3 (Desencriptación)**: En esta fase, el cliente, tras recibir el conjunto de datos evaluado, los desencripta y obtiene solo la intersección entre ambos conjuntos.

  Teniendo esto en cuenta, se han realizado pruebas modificando el tamaño de los conjuntos para ver como afecta esta variable temporalmente a los diferentes protocolos PSI.

Cabe destacar que se han tomado medidas y hecho pruebas que al final se han descartado debido a que no apartaban información adicional. Durante las pruebas de PSI se modificó también la longitud de la clave que empleaban los criptosistemas empleados, pero los datos y conclusiones obtenidos eran similares que los obtenidos en el análisis de los dichos criptosistemas.

Además, tampoco se ha visto relevante mostrar los resultados obtenidos de la Intersección Privada de Conjuntos que solo devuelve la cardinalidad del conjunto. Como es de esperarse, la fase de desencriptación reduce su tiempo considerablemente, así como la de evaluación. Sobre todo en implementaciones de PSI como la basada en *Bloom Filters* porque tras la desencriptación es necesario usar la estructura de datos para comprobar por cada elemento del conjunto si está contenido en dicho *Bloom Filter*.

4.4.1. Evaluación Resultados Criptosistemas

En esta sección nos centramos en analizar y examinar los resultados obtenidos durante las pruebas realizadas para los criptosistemas de Paillier y Damgard-Jurik.

Como ya se ha comentado anteriormente, las pruebas se han realizado variando la longitud de la clave para los criptosistemas. Además, se han escogido diferentes valores de s para el criptosistema de Damgard-Jurik. Los resultados obtenidos han sido los siguientes:

- **Resultados Tiempos de Generación de Claves**.

  En la gráfica 4 observamos que a medida que aumenta el tamaño de la clave aumenta, el tiempo de generación es también mayor. Esto se debe a que los números primos con los que se trabajan en estos criptosistemas son cada vez mayores, lo que hace que la computación necesaria para encontrar dichas claves sea mayor.

  Además, observamos que el criptosistema de Paillier, obtiene tiempos de generación de claves menores que el de Paillier, incluso en el caso de $s = 1$ en el que se trabaja sobre la misma base en el espacio de cifrados $Z_n{}^2$. Esto es seguramente debido a la necesidad a que la computación de g en Damgard-Jurik precisa de pasos adicionales (Encontrar j y k válidos).

  El incremento del tiempo de generación de las claves al aumentar s es normal, ya que se está trabajando en un espacio más grande $\mathbb{Z}_{n^{s+1}}$ por los números relacionados con las operaciones son mayores.

- **Resultados Tiempos de Encriptación.**

  Los resultados se pueden observar en la gráfica 5. En esta se puede ver que se sigue la trayectoria del caso anterior, en el que Damgard-Jurik es más costoso que Paillier y que con el aumento de s aumenta el coste computacional. En este caso, al no haber computaciones adicionales como en la generación de claves, cuando $s = 1$ observamos que los tiempos entre Paillier y Damgard-Jurik son casi iguales.

  Los tiempos de encriptación recogidos alcanzan valores mayores que en la generación de claves. Esto es seguramente debido a que en la encriptación es necesaria aplicar la exponenciación modular, siendo una de las partes computacionalmente más costosas del sistema.

- **Resultados Tiempos de Desencriptación.**

  Los resultados se pueden observar en la gráfica 6. Observamos que la tendencia que siguen los datos recopilados es la misma que en los casos anteriores. No obstante, los tiempos de ejecución son menores. Esto es gracias a que se precomputan ciertas variables y se almacenan para su uso (*mu* y *lambda).*

- **Resultados Tiempos de Adición Homomórfica.**

  En la gráfica 7 se pueden observar los datos obtenidos de las pruebas. En general, para todas las operaciones homomórficas soportadas se observa que cuando $s = 1$ los tiempos entre Paillier y Damgard-Jurik son casi iguales. Se sigue la misma tendencia en la que claves de mayor tamaño se traducen en mayores tiempos de cómputo, pero con valores temporales menores, aunque la tendencia es la misma.

- **Resultados Tiempos de Adición Entre Escalar y Texto Encriptado**

  A partir de la ilustración 8 se observa que la adición entre un escalar y un cifrado sigue la misma tendencia que la adición homomórfica.

  No obstante, los tiempos obtenidos totales son menores que en el caso de adición homomórfica. Esto es debido a que en la multiplicación que se realiza, uno de los números no está encriptado, por lo que tiene un valor menor que si estuviese encriptado. Al aumentar s el factor de expansión del criptosistema aumenta, resultando en un cifrado de mayor tamaño y mayor valor numérico, lo que se traduce en operaciones más costosas.

- **Resultados Tiempos de Multiplicación Entre Escalar y Texto Encriptado.**

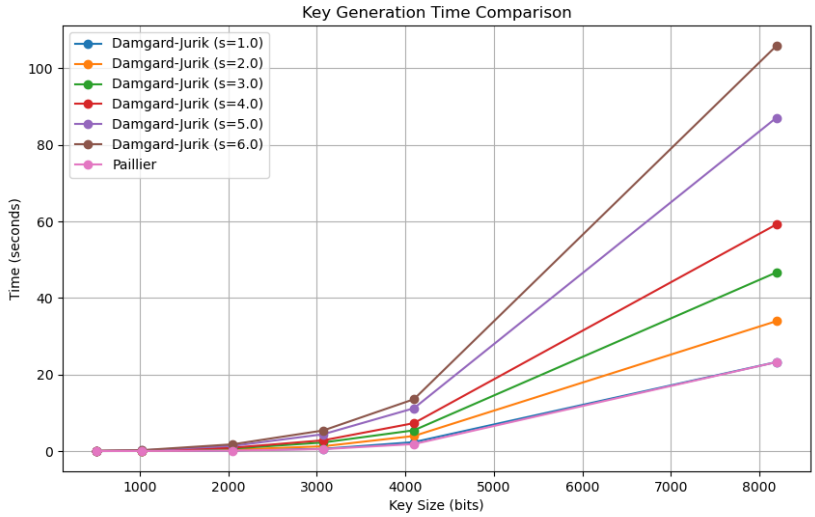

Ilustración 4: Tiempos de generación de las claves según el tamaño de la misma

En la ilustración 9 observamos la misma tendencia entre tamaño de clave y tiempo de computación de los casos anteriores.

Sin embargo, el coste temporal de la multiplicación entre un escalar con un cifrado es mayor que en el caso de la adición. Esto es debido a que en la operación realizada, ($c_{\text{mul}} = c^k$ mód n^{s+1}), se tiene que elevar el texto cifrado por el escalar. Teniendo en cuenta que el cifrado suele tener un valor numérico mayor que el texto plano y que va aumentando con el tamaño de la clave, esto tiene sentido.

A lo largo de las pruebas se ha observado que los resultados obtenidos de los criptosistemas siguen la misma tendencia. A mayor tamaño de clave, mayor coste computacional. Esto es debido a que al trabajar con números mayores a medida que aumenta el tamaño de la clave, las exponenciaciones son más caras en consecuencia. En el código se hace uso del método de *Sage power_mod()* que devuelve la exponenciación de un número en el módulo elegido para reducir el coste.

Además, el coste de computacional de Damgard-Jurik en las operaciones es mayor que el de Paillier a medida que el factor de expansión (s) va aumentando.

Cabe destacar también que las operaciones homomórficas más son rápidas en comparación con aquellas relacionadas con la generación de claves, la encriptación y, en menor medida, la desencriptación. Seguramente debido a que la aplicación de dichas operaciones es más sencilla y simple.

4.4.2. Evaluación Resultados de los Protocolos PSI

En este apartado analizamos los resultados obtenidos de computar PSI en conjuntos de datos de diversos tamaños. Como se comentó anteriormente, se hicieron más pruebas modificando otras variables, como el tamaño de clave, pero los resultados obtenidos no se consideraron pertinentes.

En las pruebas realizadas se ha modificado el tamaño máximo de los conjuntos de cada participante del PSI y se han recopilado datos temporales de las tres fases delimitadas del PSI.

El tamaño de clave que se ha elegido por defecto para todas las claves ha sido 2048, con un valor $s = 2$ para Damgard-Jurik. El tamaño de la clave se ha escogido siguiendo las recomendaciones de esta página. Según esta, para los criptosistemas desarrollados, es seguro emplear durante 2019-2030 una clave con longitud 2048. Los resultados obtenidos han sido los siguientes:

- **OPE Private Set Intersection**

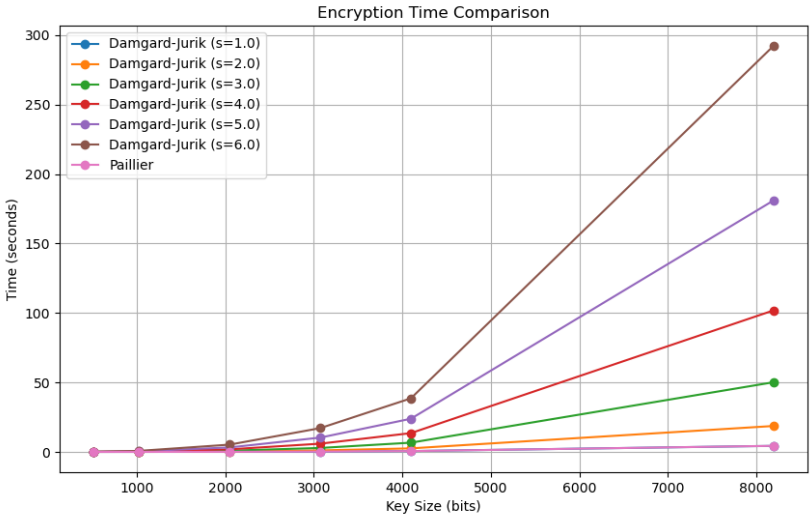

Ilustración 5: Tiempos de encriptación de un texto según el tamaño de la clave

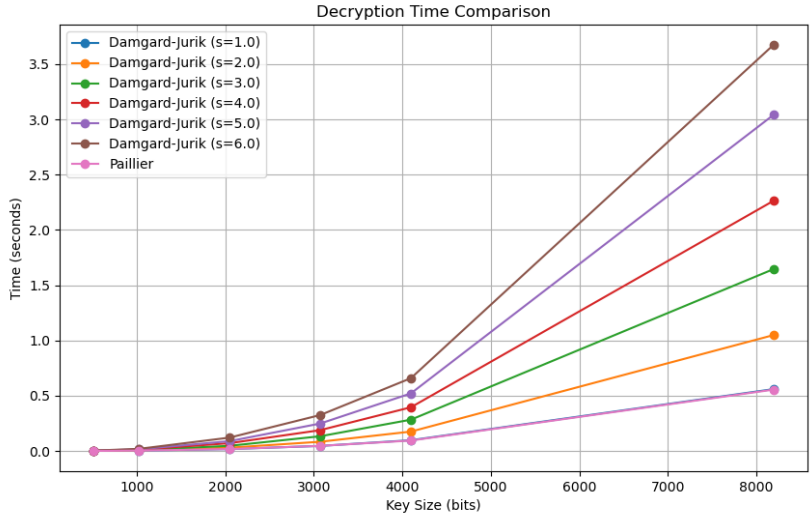

Ilustración 6: Tiempos de desencriptaicón de un texto cifrado según el tamaño de la clave

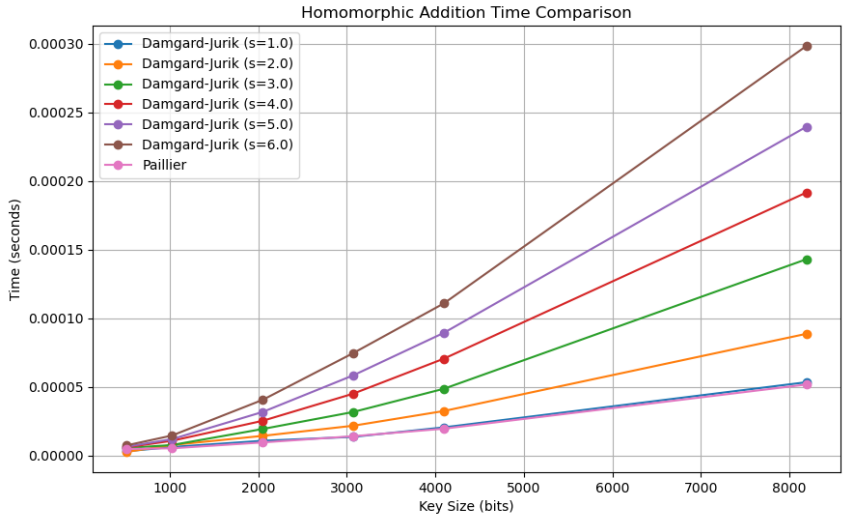

Ilustración 7: Tiempos necesarios para computar la suma de dos cifrados según el tamaño de la clave

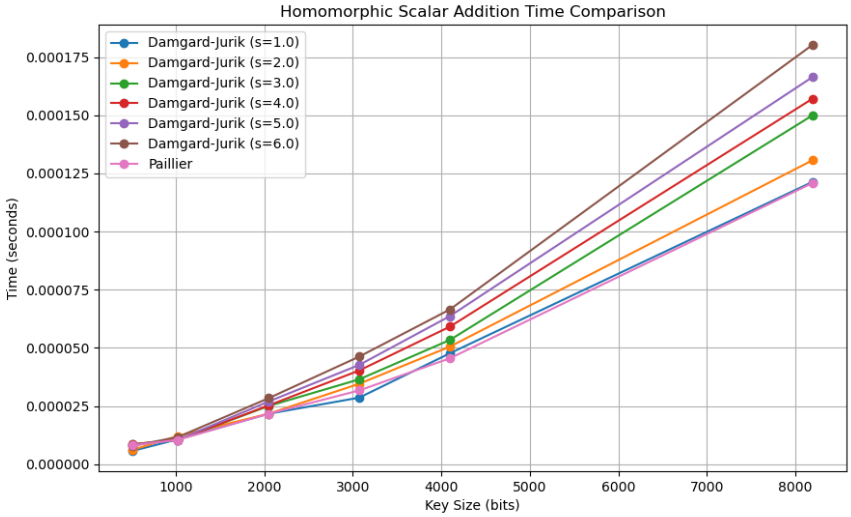

Ilustración 8: Tiempo necesario para computar la suma de dos cifrados según el tamaño de la clave

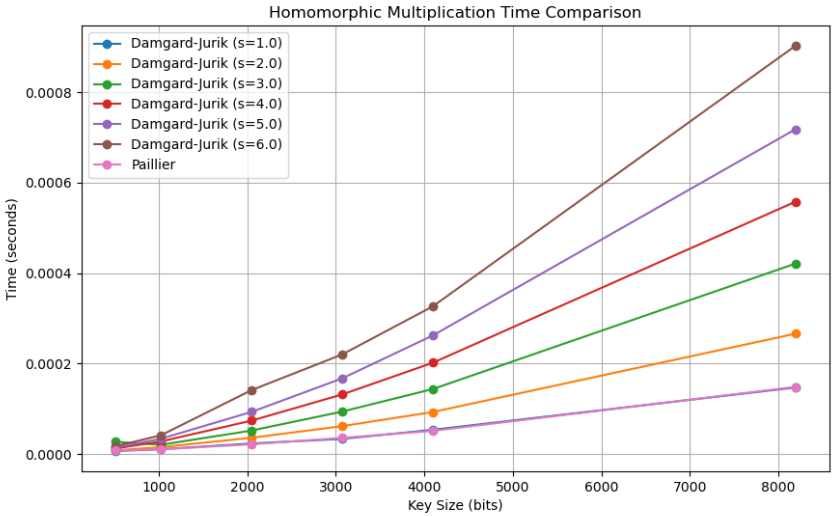

Ilustración 9: Tiempos obtenidos de aplicar la multiplicación entre un cifrado y un escalar

En primer lugar, podemos observar en las tres ilustraciones en 10 se observa una misma tendencia, en la cual a medida que el tamaño de los conjuntos va aumentando, el tiempo de cómputo de cada fase también es mayor. Esto no tiene ningún misterio y es lógico.

No obstante, en el criptosistema de Damgard-Jurik presenta un crecimiento mucho más pronunciad que el de Paillier. Al haber elegido $s = 2$ para Damgard-Jurik, el espacio sobre el que trabaja es mayor que el de Paillier. Además, observando los resultados obtenidos del análisis de los criptosistemas, esta tendencia se explica.

Cabe destacar, que a pesar de que en la fase de desencriptación el crecimiento temporal de Damgard-Jurik es superior al de Paillier, los tiempos absolutos son considerablemetne menores. Esto viene dado que la desencriptación es menos costosa que la encriptación, como pudimos observar anteriormente.

Finalmente, es interesante que a pesar de que en la fase de evaluación hay solo operaciones homomórficas (Con costes computacionales bajos), su tiempo es parecido al de la fase de la encriptación, donde se tienen que computar los coeficientes del polinomio a enviar. Esto es debido a que el servidor tiene que encriptar su propio conjunto de datos con la clave pública de su contraparte para poder evaluarlo, lo que introduce el coste adicional.

- **Bit Vector Based Private Set Intersection**

  Analizando los resultados mostrados en la ilustración 11 observamos un comportamiento similar en las fases de encriptación y desencriptación, donde Damgard-Jurik tarda más en terminar la computación que Paillier. En general, los resultados obtenidos son similares.

  En los tiempos planos sí que se observa que los costes de la fase de encriptación son bastante superiores que en el caso de OPE PSI. Esto es debido a que en el caso de OPE PSI solo es necesario encriptar los coeficientes del polinomio obtenido a partir de su conjunto de datos.

  No obstante, en el protocolo actual es necesario encriptar todo el dominio de datos, ya que se representan los conjuntos mediante vectores de bits. En las pruebas realizadas, el dominio de los datos se ha definido como tres veces superior al tamaño máximo de los conjuntos.

  Sin embargo, aunque el coste de encriptación sea menor, el caso de uso presentado en el artículo que lo introduce Ruan et al. [2019] se expone que los conjuntos son precomputados, por lo que solo añadiría coste temporal esta fase una vez.

Finalmente, en el caso de la fase evaluación observamos que los tiempos de ejecución son altos, a pesar de que el servidor no tiene que elevar el valor cifrado por su propio conjunto de datos encriptado. Esto es debido a que en la operación de evaluación definida:

$$\forall b_{Si} \in S \quad e_i = (c_{Ci})^{b_{Si}} \cdot \mathrm{En}^{pk_C}(0)$$

, a pesar de que se eleva el valor cifrado por 1 ó 0 (b_{Si}), para no desvelar información adicional en la evaluación es necesario encriptar el número 0 cada vez con la clave pública del cliente y sumarlo al resultado.

- **Bloom filter Based Private Set Intersection**

  Observando y analizando las gráficas representadas en la ilustración 12, vemos que la tendencia que siguen los datos es la misma que en los casos anteriores.

  Sin embargo, observamos un crecimiento significativo en los tiempos planos aún con tamaños de conjuntos de datos menores, tanto para la encriptación como en la desencriptación. El crecimiento es incluso mayor que en la implementación del protocolo Bit Vector PSI.

  Estos altos tiempos de encriptación y desencriptación de los conjuntos de datos vienen dados por la estructura de datos introducida en el protocolo: *Bloom Filter*. Esto viene dado a la naturaleza de esta estructura de datos y a como la hemos implementado.

  Para conseguir un *ratio* de falsos positivos de 0.01, la cardinalidad del *vector de bits* almacenado es considerablemente superior al que se almacena en el Bit Vector PSI. Además, en la desencriptación, para comprobar si un elemento se encuentra en el *Bloom Filter* es necesario computar los *hashes* del mismo, añadiendo más coste computacional, aunque este cómputo solo es necesario hacerlo una vez y almacenar el resultado.

  No obstante, observamos una reducción considerable de la fase de evaluación en comparación con las otras implementaciones de PSI. Esto es debido a que, no es necesario encriptar por parte del servidor su conjunto de datos en la operación de evaluación:

$$E(B_{FC}) \cdot g^{B_{FS}} \cdot E(-2)^r$$

  , en vez de eso, realiza una suma entre un elemento encriptado y un escalar, operación soportada tanto por Paillier como por *Damagard Jurik*. Es verdad que es necesario calcular el inverso multiplicativo de 2 y multiplicarlo por dicha suma, pero es menos costos que tener que realizar encriptaciones reiteradas sobre todo un conjunto de datos.

  Finalmente, se observa también que no se observa un crecimiento constante tan claro en los tiempos de encriptación en la fase de evaluación como en los protocolos PSI anteriores. Esto se va a atribuir a que los conjuntos de datos de las pruebas realizadas son bastante pequeños, por lo que no hay un crecimiento tan claro para esta fase, donde los costes computacionales no son tan altos.

A partir de los resultados obtenidos podemos llegar a la conclusión de que computacionalmente es más costoso el uso del criptosistema de Damgard-Jurik que el de Paillier como primitiva de los diferentes protocolos PSI implementados. Si bien es verdad que Damgard-Jurik nos permite aumentar el orden de expansión de los textos cifrados sin tener que modificar el tamaño de la clave.

En las pruebas se ha podido observar que a medida que aumenta el tamaño de los conjuntos aumenta también el tiempo de ejecución del PSI, como es lógico. Además, ha sido posible analizar las fases y operaciones de mayor coste, siendo la encriptación la operación más costosa.

Asimismo, se ha visto como se pueden evitar los costes temporales de algunas fases en la implementación de *Bloom Filter* PSI, eliminando las encriptaciones innecesarias, pero no sin un coste adicional considerable por tener que gestionar la nueva estructura de datos.

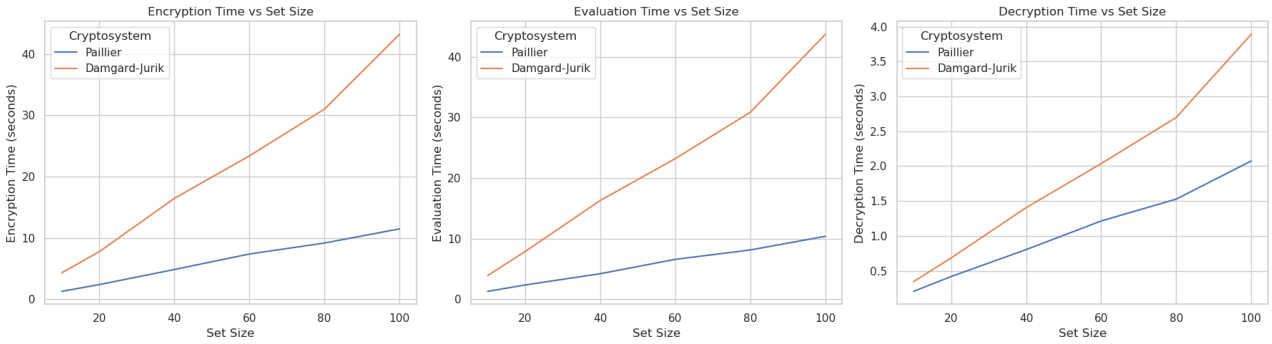

Ilustración 10: Tiempos de computación en función del tamaño de los conjuntos en OPE PSI

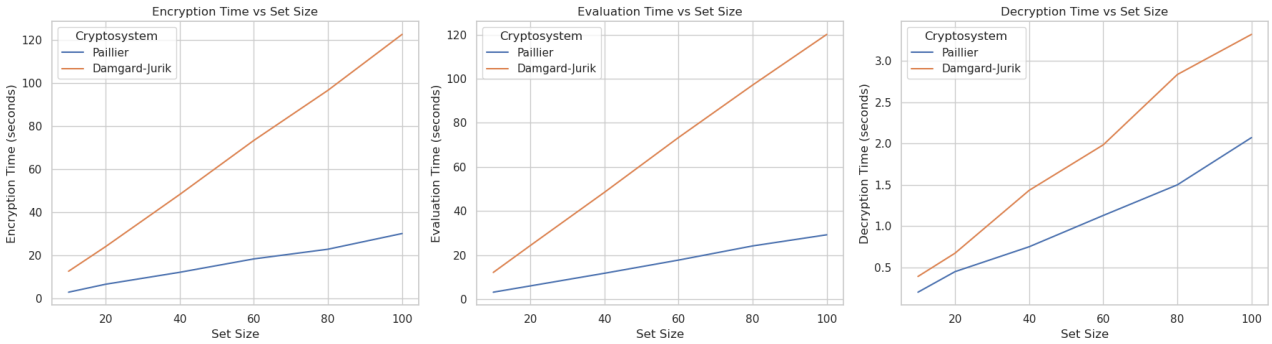

Ilustración 11: Tiempos de computación en función del tamaño de los conjuntos en Bit Vector PSI

En general, todas las implementaciones de PSI presentadas tienen sus ventajas y desventajas, con sus respectivos casos de uso. Por ejemplo, la implementación de OPE PSI viene mejor para casos en los que no se sepa a ciencia cierta exactamente el tamaño del dominio.

En el caso de BS PSI está más dirigido a computación en la nube y aplicaciones en los que el dominio del sistema sea conocida. Además, esconde la cardinalidad de los conjuntos de los participantes del PSI naturalmente gracias a que todos los conjuntos tienen el mismo tamaño.

Finalmente, BF PSI tiene casos de uso parecidos al BS PSI y protege de adversarios maliciosos con una capa de seguridad extra gracias al *Bloom Filter*. Su caso de uso sería más adecuado para protocolos O-PSI, en los que se busca delegar la fase de evaluación a una entidad externa.

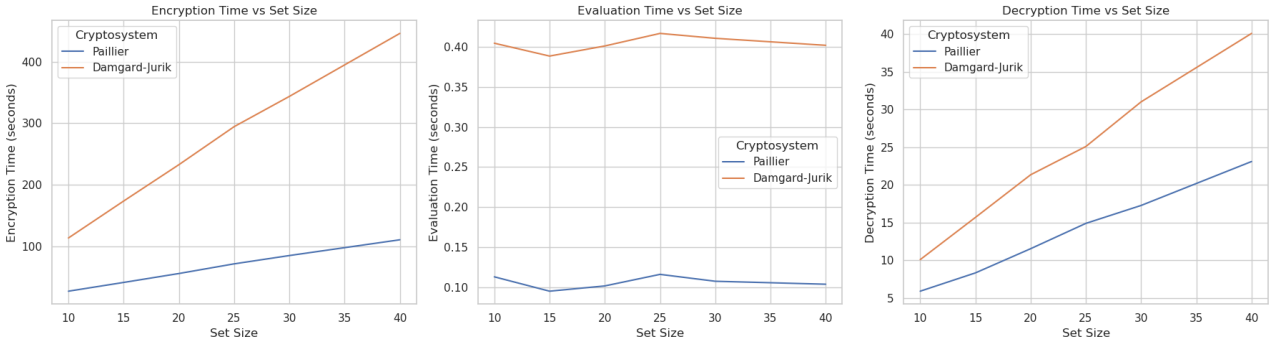

Ilustración 12: Tiempos de computación en función del tamaño de los conjuntos en Bloom Filter PSI

5 Conclusiones y Trabajos Futuros

En este capítulo queremos resumir los resultados obtenidos en el proyecto, así como posibles trabajos futuros. El *framework* que se ha presentado es fácil de extender y las implementaciones iniciales de protocolos PSI se han mostrado satisfactorias. Además, puede ser la base para futuros trabajos. Los campos que se pueden proyectar y estudiar del protocolo PSI son variados y abundantes. Se podría incluso no centrarse en PSI, y extender este *framework* para los protocolos conocidos como SMPC (Secure Multiparty Computation), del cual el problema del PSI es un caso particular.

5.1. Resultados obtenidos

En este trabajo se ofrece implementaciones diferentes de variantes de PSI (Private Set Intersection) siguiendo diversos protocolos basados en sistemas criptográficos de clave pública con propiedades homomórficas. Se han dichas implementaciones con diferentes métricas y estudiado las diferentes implementaciones PSI posibles, además de las diversas primitivas existentes y bloques de construcción.

Esto ha tenido sus dificultades y ha sido necesario estudiar y tocar solo la base matemática de la teoría de números para entender el funcionamiento de la criptografía homomórfica. Con la implementación de los protocolos PSI he sido consciente de como se pueden aprovechar las capacidades de dichos criptosistemas además de distintos casos de uso.

El estudio de los protocolos PSI me ha ayudado a comprender las diferentes implementaciones y variantes que un sistema como este puede tener según diferentes casos de uso. Además, se han podido analizar las diferentes técnicas que puede haber para preservar la privacidad en distintos contextos, como el uso de evaluación polinomial o estructuras de datos específicas, como el *Bloom Filter*.

En este proyecto se ha conseguido implementar tres de ellos de manera exitosa. El posterior análisis de los resultados obtenidos me ha ayudado en profundizar y estudiar las ventajas y desventajas de cada uno, comprendiendo los casos de uso posibles.

Con este trabajo, no solo se han alcanzado los objetivos técnicos establecidos, sino que también se ha profundizado en el conocimiento práctico y teórico necesario para abordar problemas criptográficos, obteniendo conocimiento básico para seguir investigando este tema y aplicaciones de este campo.

5.2. Trabajos Futuros

A lo largo del trabajo ya se han señalado áreas y campos que no se han tocado debido a que se encontraban fuera del alcance del proyecto. No obstante, en trabajos futuros se podrían analizar diferentes implementaciones de PSI basados en MP-PSI ó O-PSI.

Asimismo, en este trabajo solo se ha hecho uso de los criptosistemas de Paillier y Damgard-Jurik para realizar las pruebas, que son criptosistemas parcialmente homomórficos (PHE). Sin embargo, existen criptosistemas cuyo estudio sería de interés como el introducido en el artículo de Cheon et al. [2017] o el introducido por Fan y Vercauteren [2012] que introduce mejoras en el criptosistema totalmente homomórfico de Brakerski para su uso.

Además, en los distintos bloques de construcción de protocolos PSI, se han empleado estructuras de datos como el *Bloom Filter*, pero podría ser posible implementar y emplear otras de índole similar, como el *Cucko Hashing*, que es una estructura de datos para resolver las colisiones que pueda de los valores de la función *hash* en una tabla.

Finalmente, en el trabajo se ha empleado como principal primitiva de construcción la criptografía homomórfica para la construcción de protocolos PSI. No obstante, a pesar de que es la más usada, hay otras primitivas que se pueden usar, como la criptografía simétrica o los circuitos genéricos.

Referencias

J. H. Cheon, A. Kim, M. Kim, y Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi y T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

W. Diffie y M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley y D. Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

J. Fan y F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.
https://eprint.iacr.org/2012/144

M. J. Freedman, K. Nissim y B. Pinkas. Efficient private matching and set intersection. In C. Cachin y J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 1–19, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

Google. Google password checkup: Enhancing account security with cross-account protection, 2021. https://blog.google/technology/safety-security/ Septiembre 2024.

M. Green. Attack of the week: Airdrop tracing. https://blog.cryptographyengineering.com/2024/01/11/attack-of-the-week-airdrop-tracing/, January 2024. Accessed: Julio 2024.

A. Inc. Child sexual abuse material (csam) detection technical summary, 2021. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf Septiempre 2024.

D. Morales, I. Agudo y J. Lopez. Private set intersection: A systematic literature review. *Computer Science Review*, 49:100567, 2023.
https://www.sciencedirect.com/science/article/pii/S1574013723000345

P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, 5:223–238, 05 1999.

R. L. Rivest, A. Shamir y L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
http://dblp.uni-trier.de/db/journals/cacm/cacm21.html#RivestSA78

D. Robinson. *An Introduction to Abstract Algebra*. De Gruyter textbook. Walter de Gruyter, 2003.
https://books.google.es/books?id=Yj3ApD8TeCUC

K. H. Rosen. *Elementary Number Theory and Its Applications*. Addison-Wesley, 6th edition, 2011.

O. Ruan, Z. Wang, J. Mi, y M. Zhang. New approach to set representation and practical private set-intersection protocols. *IEEE Access*, 7:64897–64906, 2019.

Signal Foundation. Technology preview: Private contact discovery. https://signal.org/blog/contact-discovery/, May 2018. Accessed: Julio 2024.

X. A. Wang, F. Xhafa, X. Luo, S. Zhang, y Y. Ding. A privacy-preserving fuzzy interest matching protocol for friends finding in social networks. *Soft Computing*, 22(8):2517–2526, 2018. https://doi.org/10.1007/s00500-017-2506-x