Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática (GII)

Trabajo Fin de Grado

Aplicación Resiliente para la Agregación de Almacenamientos en la Nube.

Autor: Luis Vicente Arévalo
Tutor(a): Jorge Dávila Muro

Madrid, 06 2023

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

Trabajo Fin de Grado
Grado en Ingeniería Informática (GII)

Título: Aplicación Resiliente para la Agregación de Almacenamientos en la Nube.

06 2023

Autor: Luis Vicente Arévalo
Tutor: Jorge Dávila Muro
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El objetivo de este proyecto, es desarrollar una aplicación que combine varias cuentas de almacenamiento en la nube, con la intención de que el usuario pueda almacenar sus datos en la nube de manera cifrada, repartiéndolos entre estas. El usuario debe tener acceso a estos espacios de almacenamiento y haberlos vinculado, para que la aplicación pueda hacer uso de ellos.

En el presente documento se explicará al método empleado para aumentar la disponibilidad de los datos, la división de secretos. Además, se presentará la base matemática sobre la que se sustenta el desarrollo de este método. Acto seguido se introducirá la librería que se ha creado con el objetivo de aplicar la división de secretos a secuencia de bytes extensas.

Por último se detalla la aplicación, objetivo de este Trabajo Fin de Grado, explicando su funcionamiento y características.

Códigos UNESCO:

- **Campos:** Matemáticas (Código 12) y Ciencias Tecnológicas (Código 33)
- **Disciplinas:** Ciencia de los ordenadores (Código 1203) y Tecnología de los ordenadores (Código 3304)
- **Subdisciplinas:** Informática (Código 1203.17) y Dispositivos de almacenamiento (Código 3304.18)

Palabras clave: Almacenamiento en la nube, Ciberseguridad, División de secretos, Librería, Aplicación

Abstract

The purpose of this project, is to develop an application that combines multiple cloud storage accounts, with the objective of allowing the user to store its data in the cloud encrypted and shared between the accounts. The user must have access to this storage spaces and must have linked them with the application, so it can make use of them.

In this document it will be explained the method used to increase de availability of the data, the secret sharing schemes. It will also be presented the mathematics in which the development of this method is based. Thereafter, it will be introduced the library that has been developed with the intention of employing a secret sharing scheme over extensive byte secuences.

Finally it will be detailed the application, which is the objective of this Final Degree Project, explaining its operation and characteristics.

Keywords: Cloud storage, Cybersecurity, Secret sharing schemes, Library, Application

Tabla de contenidos

1. Introducción **1**
1.1. Estado del arte 2

2. División de secretos **7**
2.1. Esquema de Shamir 7
2.2. Aritmética modular 9
2.3. Librería de división de secretos 13

3. Aplicación Desarrollada **21**
3.1. Almacenamiento en la nube 22
3.2. Funcionamiento de la aplicación 24
3.2.1. Subida de ficheros 24
3.2.2. Descarga de ficheros 30
3.2.3. Borrar ficheros 34
3.3. Base de Datos 35
3.3.1. Protección Base de Datos 37

4. Resultados y conclusiones **43**
4.1. Líneas futuras 44

5. Análisis de impacto **47**

Bibliografía **49**

Índice de figuras

2.1. Ejemplo de la división de secretos de Shamir. 8
2.2. Recuperación de secreto con interpolación polinómica de Lagrange. 9
2.3. Ejemplos de suma y multiplicación en aritmética modular. 10
2.4. Ejemplos de multiplicación binaria con reducción. 12
2.5. Tiempo en procesar $1.28e8$ bytes ($122,07$ MB) 14

3.1. Pantalla selección fichero 24
3.2. Pantalla para configurar la división de secretos 25
3.3. Diagrama lógico seguido para la subida de un fichero 30
3.4. Pantalla en la que se selecciona el fichero a descargar 31
3.5. Diagrama lógico seguido para la descarga de un fichero 34
3.6. Diagrama E/R de la base de datos 36
3.7. Estructura fichero base de datos 39
3.8. Esquema de claves para proteger la base de datos 40
3.9. A la izquierda diagrama lógico seguido para la creación de las claves. A la derecha diagrama lógico seguido para el descifrado de la base de datos 41

Índice de cuadros

2.1. Secuencia de bytes a tratar. 13
2.2. Secuencia de bytes dividida en bloques. 15
2.3. Bloques almacenado en estructuras bnm. 16
2.4. Primer bloque desplazado un bit a la izquierda. 16
2.5. Polinomio irreducible para GF($2^{8\cdot128}$). 16
2.6. Ejemplos de coeficientes empleados para generar fragmentos. . . . 17
2.7. Secuencia de bytes dividida en bloques. 18

3.1. Nubes con soporte en la aplicación 22
3.2. Secuencia de bytes del fichero a subir. 25
3.3. Clave de cifrado del fichero y valor incial 27
3.4. Hash del fichero 27
3.5. Fichero cifrado 27
3.6. Nombres de los fragmentos 28
3.7. Fragmento 1 del fichero a subir. 28
3.8. Hash del fragmento 1 del fichero a subir. 29
3.9. Errores que se pueden producir de la descarga y recuperación de un fichero 33
3.10. Atributos almacenados en la base de datos 37
3.11. Salt derivación de claves. 39
3.12. Resultado de la derivación de claves 39
3.13. Ejemplo de valor inicial 39
3.14. Hash de la clave de cifrado 40
3.15. Cabecera base de datos 40

Capítulo 1

Introducción

La computación en la nube [1] consiste en un conjunto de servicios software cuyo uso se da en remoto. Estos servicios surgieron en la década de los noventa y con el tiempo ha ido adquiriendo popularidad e importancia. De entre todos los servicios que componen la computación en la nube, uno de los más empleados es el "almacenamiento en la nube" [2], con el que se ofrece espacio de almacenamiento remoto a los usuarios.

Por otro lado, la división de secretos [3] es una serie de métodos empleados para la distribución de un secreto dentro de un grupo, de tal manera que ninguna de las partes puede recuperar el secreto de manera individual. Además, estos esquemas permiten la recuperación del secreto a partir de un subconjunto del total de las partes. Estos esquemas fueron inventados en 1979 por Adi Shamir y George Blakley.

Este Trabajo Fin de Grado tiene como objetivo desarrollar una aplicación la cuál permita el almacenamiento de datos en la nube, aumentando su disponibilidad respecto a otros almacenamientos en la nube. Además, la aplicación tiene que asegurar que solo el propietario puede acceder a los datos que se guardan mediante la aplicación. Esta aplicación tiene que proporcionar estas funcionalidades al usuario, siempre que el dispositivo en el que se ejecuta la aplicación tenga acceso a internet y pueda tanto subir como descargar datos de distintos espacios de almacenamiento.

Para conseguir estos objetivos, la aplicación es un agregador de nubes, es decir, toma múltiples servicios de almacenamiento en la nube y los gestiona como una única nube de almacenamiento, de cara al usuario. A su vez, para aumentar la disponibilidad de los datos del usuario, el agregador hace uso de la división de

[1] (2023) Wikipedia - Computación en la nube. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing

[2] (2023) Wikipedia - Almacenamiento en la nube. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_storage

[3] (2023) Wikipedia - División de secretos. [Online]. Available: https://en.wikipedia.org/wiki/Secret_sharing

secretos.

La división de secretos divide los datos del usuario en varios fragmentos y permite recuperar los datos originales a partir de un subconjunto de todos los fragmentos. Por este motivo, se ha utilizado este método para aumentar la disponibilidad.

De esta manera, la aplicación divide los datos y guarda los fragmentos resultantes en distintas nubes. Para la recuperación de los datos, el agregador tiene que descargar una cantidad de fragmentos igual o inferior a los generados en la subida. Así se consigue recuperar los datos sin tener que acceder a todas las nubes en las que están almacenados.

Adicionalmente, los fragmentos generados mediante la división de secretos no proporcionan información alguna acerca de los datos que se dividen. Esta propiedad asegura que el proveedor del servicio de almacenamiento no tiene acceso a información alguna acerca de los datos que el usuario sube. No obstante, para asegurar que únicamente el usuario puede recuperar los datos almacenados, la aplicación cifra dichos datos antes de aplicar la división de secretos.

La aplicación funciona de manera local, es decir, no cuenta con un servidor activo al que el usuario mande los datos. En su lugar, los datos se procesan en el dispositivo del usuario y se mandan directamente a la cuenta correspondiente. Esta decisión se ha tomado para evitar problemas de disponibilidad del servicio que pudieran surgir al emplear un servidor y que la conexión a este fallase.

El desarrollo de este agregador se ha dividido en dos secciones principales. La primera de ellas ha sido el desarrollo de una librería que aplique la división de secretos a un fichero, entendiendo como fichero una unidad de almacenamiento lógica que agrupa un conjunto de datos relacionados entre sí. Esta librería tiene que permitir la división de un fichero en múltiples fragmentos, y la recuperación del fichero original a partir de un subconjunto del total de los fragmentos.

La segunda sección principal de este proyecto ha consistido en el desarrollo de la aplicación, como intermediaria entre los usuarios y distintos servicios de almacenamiento en la nube. Los servicios de almacenamiento en la nube empleados por la aplicación son distintas cuentas, en este tipo de servicios, dadas por el usuario a la aplicación. Estas cuentas se emplean por la aplicación para almacenar los distintos fragmentos, que son el resultado de aplicar la librería a los ficheros que se quieren guardar.

1.1. Estado del arte

El almacenamiento en la nube, como ya se ha comentado, es uno de los servicios en la nube más populares. Esto ha llevado a que existan una gran cantidad de proveedores de este tipo de servicios. No obstante, en la mayoría de casos, estos proveedores guardan los datos de sus usuarios, sin atender a varios conceptos de seguridad.

Por ejemplo, la integridad de la información que sube un usuario no se comprueba en la mayoría de nubes. Esto da paso a que el usuario pueda descargar de la nube datos que no coincidan con los que se guardaron en su momento.

Otra gran olvidada en los servicios de almacenamiento en la nube es la disponibilidad. Muchos de los proveedores almacenan los datos de un usuario en un único servidor, por lo que si el usuario no puede establecer una conexión con este servidor, no puede recuperar su datos. Por otro lado, también existen proveedores como Google, los cuales dividen los datos de los usuarios en secciones, y reparten estas secciones entre distintos servidores. Sin embargo, la incapacidad de recuperar alguna de estas secciones, supone que el usuario no pueda recuperar la información.

La confidencialidad de los datos, es la única protección que los proveedores de servicios en la nube, suelen dar a los ficheros de sus usuarios. Para ello, la práctica más común es cifrar estos datos con una clave única, generada por el proveedor. Esto implica que una vez los datos se encuentran en la nube, el proveedor es el único capaz de obtenerlos en claro, mientras que el usuario tiene que solicitar al servicio que se los descifre y se los devuelva en claro.

Frente al cifrado con claves generadas y conocidas por el proveedor, existen servicios de almacenamiento en la nube, como Mega o PCloud, los cuales emplean cifrado por parte del cliente. Este tipo de cifrado genera las claves de cifrado a partir de la contraseña del cliente. Además, cifra los datos en local antes de guardarlos en la nube, es decir, en ningún momento el proveedor cuenta con los datos en claro. No obstante, esta clase de servicios de almacenamiento en la nube, en los que el usuario es el único capaz de recuperar sus datos en claro, son la minoría de entre todos los existentes.

Todas las nubes de almacenamiento cuentan con su propio servicio para acceder al espacio de un usuario, y los contenidos guardados en él. Además, varios de estos proveedores incluyen APIs propias, a partir de las cuales poder acceder a sus servicios. No obstante, apenas existen softwares que permitan emplear las nubes de varios de estos proveedores para almacenar los datos de sus usuarios. Es decir, apenas existen agregadores de nubes públicas.

Existen softwares que agreguen diferentes espacios de almacenamiento. La mayoría de estos están diseñados para ser usados dentro de una organización para emplear varios servidores privados como espacio de almacenamiento. Sin embargo, también existen softwares que permiten el incluir múltiples espacios de servicios en la nube, como un único sistema de ficheros. De estos, el más destacado es rclone [4]. Rclone permite a sus usuarios montar distintos espacios de almacenamiento en la nube como directorios dentro de un sistema de ficheros.

Los esquemas de división de secretos, fueron inventados, de manera independiente, por Adi Shamir y George Blakley en 1979, dando lugar al esquema de división de secretos de Shamir y al esquema de división de secretos de Blakley.

[4](2023) Rclone. [Online]. Available: `https://rclone.org/`

Ambos esquemas se definen por un par de valores númericos (k, n), donde el primero de estos valores (k) es el valor límite (*threshold*), el cuál indica la cantidad mínima de fragmentos necesarios para recuperar el fichero original, y el segundo valor (n), indica la cantidad de fragmentos que el esquema tiene que generar.

Por otro lado, la diferencia entre ambos esquemas es que el esquema de división de secretos de Shamir [5] se basa en la interpolación Polinómica de Lagrange [6], mientras que el de Blakley [7], emplea la intersección entre planos no paralelos. En cuanto al uso que se da de los dos esquemas, el más empleado es el esquema de Shamir, debido a que es el más eficiente de los dos en cuanto al tamaño de los fragmentos. En el esquema de división de secretos de Shamir los fragmentos son tan grandes como el secreto original, mientras que en el esquema de Blakley son k veces más grandes.

Acerca del uso que se da a estos esquemas, apenas existen softwares que empleen ninguno de los dos, quedando casi exclusivamente relegados para proteger claves cortas que se quieren distribuir entre varias partes. El principal ejemplo de software que emplea la división de secretos consiste en *ssss* [8]. Este software emplea el esquema de división de secretos de Shamir (*Shamir's Secret Sharing Scheme*) para dividir una cadena de hasta ciento veintiocho (128) caracteres ASCII. Esta secuencia de caracteres, o bytes, se le pasan por la entrada estándar, escribiendo los fragmentos resultantes en la salida estándar. A pesar de esto, no parecen existir softwares que apliquen la división de secretos a secuencias de bytes más largas como es el caso de ficheros, o que implementen la división de secretos de Blakley.

Aún con los esquemas de división de secretos relegados principalmente a la división de pequeñas cadenas de caracteres, como pueden ser contraseñas, existen softwares con la capacidad de repartir ficheros entre varios servidores, y de recuperar el fichero original a partir de la información almacenada en un subconjunto del total de los servidores. No obstante, en estos casos en lugar de emplear un esquema de división de secretos, se emplean esquemas de corrección de errores similares a RAID5 [9]. Un ejemplo de estos softwares es *zfec* [10].

A modo de resumen, para cada fichero, se divide éste en tantas partes como servidores mínimos son necesarios para recuperar el fichero. Tras fragmentar el fichero, las copias extras que se quieren generar se producen como códigos de redundancia, o de corrección de errores, de los fragmentos del fichero original. Si bien la corrección de errores permite aumentar la disponibilidad de los datos que se guardan, los fragmentos resultantes dan información acerca del fiche-

[5](2023) Wikipedia - Esquema de División de Secretos de Shamir. [Online]. Available: `https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing`

[6](2023) Wikipedia - Interpolación Polinómica de Lagrange. [Online]. Available: `https://en.wikipedia.org/wiki/Lagrange_polynomial`

[7](2023) Wikipedia - Esquema de División de Secretos de Blakley. [Online]. Available: `https://en.wikipedia.org/wiki/Secret_sharing#Blakley's_scheme`

[8](2023) ssss. [Online]. Available: `http://point-at-infinity.org/ssss/`

[9](2023) RAID5. [Online]. Available: `https://www.ionos.es/digitalguide/servidores/seguridad/raid-5/`

[10](2023) zfec. [Online]. Available: `https://tahoe-lafs.org/trac/zfec/`

ro original, cosa que no se produce si se emplean los esquemas de divisón de secretos.

A pesar de esto, los esquemas de corrección de errores son más populares que los de división de secretos. Los esquemas de corrección de errores son la principal solución empleada para aumentar la disponibilidad de datos. El motivo de esto es que son más eficientes que los esquemas de división de secretos, tanto en el tiempo necesario para calcular todos los fragmentos como en el tamaño de los fragmentos.

Mientras que en los esquemas de división de secretos es necesario calcular los n fragmentos, en los esquemas de corrección de errores solo es necesario calcular $n - k$ fragmentos, ya que los k primeros fragmentos están conformados por los datos a fragmentar. Por otro lado, mientras el esquema de Shamir crea fragmentos del mismo tamaño que los datos originales, los fragmentos que crean los esquemas de corrección de errores ocupan $1/k$ de lo que ocupan los datos divididos.

Capítulo 2

División de secretos

En este apartado se van a explicar el algoritmo de división de secretos empleado en la aplicación. Este es el esquema de división de secretos de Shamir [1], ya que como se ha comentado en el apartado Estado del arte, es el más eficiente. Adicionalmente, se explica la librería de división de secretos desarrollada para la aplicación.

2.1. Esquema de Shamir

El esquema de Shamir o esquema de división de secretos de Shamir, es un esquema de división de secretos (*k*,*n*) basado en la idea de que **solo se puede obtener un único polinomio de grado** $k-1$ **a partir de *k* puntos que pertenecen al polinomio**[2]. De esta manera, cuando se quiere dividir un secreto, el esquema genera un polinomio de grado $k-1$ (ecuación 2.1). Los coeficientes de este polinomio se generan de manera aleatoria, a excepción del coeficiente constante, al cuál se le asigna el valor del secreto a fragmentar.

$$S \equiv Secreto$$
$$\{a_{k-1}, \ldots, a_1\} \leftarrow \text{Valores aleatorios}$$

$$f(x) = a_{k-1} \cdot x^{k-1} + ... + a_2 \cdot x^2 + a_1 \cdot x + S \tag{2.1}$$

Una vez se ha generado el polinomio, el esquema de Shamir genera los *n* distintos fragmentos a partir de este polinomio. Cada uno de estos fragmentos cuenta con un **índice propio, el cuál se tiene que guardar con el fragmento para poder recuperar el valor original**.

Además, estos fragmentos se corresponden con el valor de los puntos del polinomio $f(1), f(2), \ldots, f(n)$. De esta manera el fragmento 1 es el punto $f(1)$, el

[1] A. Shamir, "How to share a secret," Commun. ACM, vol. 22, no. 11, p.612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[2] (2023) Wikipedia - Teorema de la interpolación. [Online]. Available: https://en.wikipedia.org/wiki/Polynomial_interpolation#Interpolation_theorem

fragmento 2 es $f(2)$ y así sucesivamente hasta obtener los *n* fragmentos. El punto $f(0)$ se correspondiendo así con el secreto que se quiere fragmentar.

Por ejemplo, si se quiere dividir el valor ciento once (111) en cinco (5) fragmentos, y además se quiere recuperar este valor con solo tres (3) de estos fragmentos, se crea un polinomio de grado dos. Los coeficientes de este polinomio se obtienen de manera aleatoria, a excepción del coeficiente constante, el cuál toma el valor a fragmentar. La creación de los fragmentos se presenta en la figura 2.1.

$$
\begin{aligned}
&\text{Secreto } = 111 \\
&[a_1, a_2] = [150, 83] \text{ (Valores aleatorios)} \\
&f(x) = 83 \cdot x^2 + 150 \cdot x + 111 \\
&\text{Fragmento } 1 = f(1) = 83 + 150 + 111 = 344 \\
&\text{Fragmento } 2 = f(2) = 83 \cdot 4 + 150 \cdot 2 + 111 = 743 \\
&\text{Fragmento } 3 = f(3) = 83 \cdot 9 + 150 \cdot 3 + 111 = 1308 \\
&\text{Fragmento } 4 = f(4) = 83 \cdot 16 + 150 \cdot 4 + 111 = 2039 \\
&\text{Fragmento } 5 = f(5) = 83 \cdot 25 + 150 \cdot 5 + 111 = 2936
\end{aligned}
$$

Figura 2.1: Ejemplo de la división de secretos de Shamir.

Para la recuperación del secreto, el esquema de Shamir recupera *k* fragmentos cualesquiera del total. El secreto se obtiene a partir de la interpolación de los *k* fragmentos recuperados, para obtener el punto $f(0)$. Cualquier método de interpolación permite la obtención de un polinomio único a partir de varios puntos pertenecientes a este, y con ello el cálculo de cualquier otro punto perteneciente a dicho polinomio.

Más concretamente, la interpolación permite obtener un polinomio de grado k a partir de $k + 1$ puntos. Para estos cálculos normalmente se emplea la Interpolación Polinómica de Lagrange[3], dado que el tiempo para la obtención de los coeficientes del polinomio es lineal, sin embargo, cualquier otro método de interpolación es válido.

En la figura 2.2 se muestra la recuperación del secreto fragmentado en la figura 2.1. Este secreto se recupera a partir de tres fragmentos cualesquiera de los cinco generados. En el caso del ejemplo, se recupera el secreto a partir de los fragmentos con los índices 1, 2 y 4, empleando la Interpolación Polinómica de Lagrange.

[3](2023) Wikipedia - Interpolación Polinómica de Lagrange. [Online]. Available: `https://en.wikipedia.org/wiki/Lagrange_polynomial`

$$f(0) = f(1) \cdot (\frac{0-2}{1-2} \cdot \frac{0-4}{1-4}) + f(2) \cdot (\frac{0-1}{2-1} \cdot \frac{0-4}{2-4}) + f(4) \cdot (\frac{0-1}{4-1} \cdot \frac{0-2}{4-2})$$
$$f(0) = 344 \cdot \frac{(-2) \cdot (-4)}{(-1) \cdot (-3)} + 743 \cdot \frac{(-1) \cdot (-4)}{1 \cdot (-2)} + 2039 \cdot \frac{(-1) \cdot (-2)}{3 \cdot 2}$$
$$f(0) = 344 \cdot \frac{8}{3} + 743 \cdot \frac{4}{-2} + 2039 \cdot \frac{2}{6}$$
$$f(0) = 111$$

Figura 2.2: Recuperación de secreto con interpolación polinómica de Lagrange.

2.2. Aritmética modular

Para que la división de secretos se pueda aplicar a secuencias de bytes, es necesario emplear aritmética modular. La aritmética modular es un sistema, en el que los números pertenecen a un conjunto finito de valores. Este conjunto se denomina campo finito (F) o campo de Galois (GF), y la cantidad de valores que lo conforman se indica mediante un módulo p (F_p ó $GF(p)$), también denominado orden o tamaño.

Es necesario emplear esta aritmética, dado que la cantidad de valores que se pueden representar con una secuencia de bytes no es infinita. Por ejemplo, un único byte solo puede tomar doscientos cincuenta y seis (256) valores distintos. Esto hace que cualquier operación que se realice sobre el valor del byte, dé como resultado otro valor de entre estos doscientos cincuenta y seis posibles valores.

Las operaciones básicas de suma, resta y multiplicación, se realizan de manera similar a la aritmética sobre números naturales. Tomando la suma de dos números como ejemplo, en un campo finito de módulo p, se empieza sumando ambos números. A esta suma se le divide por el orden p. El resto de dicha división constituye el resultado de la suma en aritmética modular [4]. Las otras operaciones siguen este esquema.

En la figura 2.3, se muestra un ejemplo de la suma y la multiplicación de dos números, en el campo de Galois de noventa y siete (97).

$$GF(97)$$

$$(83 + 50) \text{ modulo } 97$$
$$133 \text{ modulo } 97$$
$$133 = 1 \cdot 97 + 36$$
$$133 \text{ modulo } 97 = 36$$

[4](2023) Wikipedia - Aritmética modular. [Online]. Available: https://en.wikipedia.org/wiki/Modular_arithmetic#Integers_modulo_n

$$(83 \cdot 3) \text{ modulo } 97$$
$$249 \text{ modulo } 97$$
$$249 = 2 \cdot 97 + 55$$
$$249 \text{ modulo } 97 = 55$$

Figura 2.3: Ejemplos de suma y multiplicación en aritmética modular.

En cuanto a la división, también denominada inversa multiplicativa, las diferencias son mayores que en el resto de operaciones básicas. En aritmética modular, un número x tiene inversa multiplicativa solo si es coprimo del módulo p, es decir, si el máximo común divisor del número x y el módulo p es uno (1).

Esta inversa multiplicativa de un número x es el valor que al multiplicalo por dicho número x en el campo finito de módulo p da como resultado uno (1) [5]. El principal algoritmo que se usa para calcular esta inversa, y por tanto el usado en la librería desarrollada, es el algoritmo extendido de Euclides[6].

Según lo explicado hasta ahora, solo los campos finitos con módulo primo tienen inversa multiplicativa para todos sus valores. Sin embargo, se puede conseguir crear campos finitos de módulo no primo, con inversa multiplicativa para todos sus valores. Para ello, es necesario incluir un polinomio irreducible[7] ($poly$) con el que operar en dicho campo[8].

Estos polinomios irreducibles suelen ser trinomios o pentanomios, es decir, están compuestos por tres o cinco coeficientes respectivamente. Se utilizan este tipo de polinomios, debido a que el tiempo necesario para operar con un polinomio irreducible, es directamente proporcional al número de coeficientes que lo componen. La razón por la que también se emplean pentanomios es porque para los campos finitos múltiplos de ocho (8), no siempre existe un trinomio que se puede emplear como polinomio irreducible [9], mientras que siempre existe un pentanomio como polinomio irreducible para estos Campos de Galois.

Por otro lado, al operar con secuencias de bytes como se ha indicado anteriormente, todas las operaciones que se realizan son a nivel de bit. Esto origina, que las operaciones de suma, resta y multiplicación se sustituyan por operaciones binarias en campos finitos [10]. Estas operaciones se realizan sobre un campo

[5](2023) Wikipedia - Inversa multiplicativa. [Online]. Available: https://en.wikipedia.org/wiki/Modular_multiplicative_inverse

[6](2023) Wikipedia - Algoritmo extendido de Euclides. [Online]. Available: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

[7](2023) Wikipedia - Polinomio irreducible. [Online]. Available: https://en.wikipedia.org/wiki/Irreducible_polynomial

[8](2023) Wikipedia - Factorización de Polinomios sobre campos finitos. [Online]. Available: https://en.wikipedia.org/wiki/Factorization_of_polynomials_over_finite_fields

[9]G. Seroussi, "Table of low-weight binary irreducible polynomials," in HP Labs Technical Reports, HP, Ed., 1998.

[10](2023) Wikipedia - Aritmética en campos finitos. [Online]. Available: https://en.wikipedia.org/wiki/Finite_field_arithmetic

finito con un polinomio irreducible, por el hecho de que así todos los valores del campo de Galois pueden tener inversa multiplicativa.

El hecho de que todos los valores del campo finito cuenten con inversa multiplicativa, es necesario para la división de secretos. Más concretamente, se necesita la inversa multiplicativa para poder recuperar el secreto a partir de los fragmentos. Esto se debe a que la interpolación de varios puntos emplea la operación de inversa multiplicativa o división. Si alguno de los valores no contara con inversa multiplicativa, no se podría realizar la interpolación, y por tanto, no se podría recuperar el secreto.

La suma a nivel de bit consiste en la operación lógica **XOR** ($\oplus$). La operación XOR es su propia inversa, es decir, si a un valor se le aplica la operación XOR consigo mismo, el resultado es cero (0). Por este motivo, además de la suma, la resta también se sustituye por la operación XOR. Esto causa que la negativa de un número sea el mismo número. Por ejemplo, se da que $-1 = 1$.

Respecto a la multiplicación, esta se sustituye por la multiplicación binaria [11]. Esta operación se compone de una combinación de desplazamientos de bits a la izquierda ($\ll$) y sumas como XORs. Cada desplazamiento de n bits a la izquierda supone multiplicar un número por la enésima potencia de dos (2^n).

A modo de resumen, este algoritmo es idéntico al algoritmo de multiplicación estándar [12]. Las únicas diferencias entre ambos algoritmos es que la multiplicación estándar se da en base diez, mientras que la multiplicación binaria se da en base dos y cambiando las sumas por XORs.

Adicionalmente, durante la multiplicación el desplazamiento de bits a la izquierda se realiza bit a bit. El objetivo de esto es reducir cualquier valor resultante (y) que se encuentre fuera del campo de Galois empleado. En los casos en los que este desbordamiento se produce, se hace uso del polinomio irreducible $poly$ del campo de Galois con el que se está trabajando. Para ello, se aplica al resultado y la operación XOR con el polinomio irreducible $poly$ ($y \oplus poly$) [13].

En la figura 2.4 se presentan dos ejemplos de la multiplicación binaria en un campo finito. Ambas multiplicaciones se dan en el campo de Galois de doscientos cincuenta y seis (256), empleando como polinomio irreducible el valor doscientos ochenta y tres (283). El primer ejemplo consiste en multiplicar el valor ochenta y tres (83) por cuatro (4). Por otro lado, el segundo ejemplo es la multiplicación entre el valor ciento ciencuenta (150) y el valor cinco (5).

[11](2023) Wikipedia - Multiplicación binaria. [Online]. Available: `https://en.wikipedia.org/wiki/Binary_multiplier#Binary_long_multiplication`

[12](2023) Wikipedia - Algoritmo de multiplicación. [Online]. Available: `https://en.wikipedia.org/wiki/Multiplication_algorithm#Long_multiplication`

[13](2023) Wikipedia - Multiplicación sobre campos finitos. [Online]. Available: `https://en.wikipedia.org/wiki/Finite_field_arithmetic#Multiplication`

$GF(256)$ con polinomio irreducible $283(0\text{x}11\text{B})$

Multiplicar por cuatro es equivalente a desplazar dos bits a la izquierda
$83 \cdot 4 = 83 \ll 2$
Desplazamos bit a bit el número dos bits a la izquierda
$83 \ll 2 = 166 \ll 1$
$166 \ll 1 = 332$
Como el valor está fuera del campo finito, se reduce con el polinomio irreducible
$332 \oplus 283 = 87$
El resultado de la operación por tanto es ochenta y siete
$83 \cdot 4 = 87$

Multiplicar por cinco es equivalente a desplazar dos bits a la izquierda y sumar por el valor original
$150 \cdot 5 = 150 \ll 2 \oplus 150$
Desplazamos un bit a la izquierda
$150 \ll 2 = 300 \ll 1$
Como el desplazamiento causa desbordamiento, se reduce con el polinomio irreducible
$300 \oplus 283 = 55$
$150 \ll 2 = 55 \ll 1$
Desplazamos otro bit a la izquierda
$55 \ll 1 = 110$
El resultado de la operación se calcula
$150 \cdot 5 = 110 \oplus 150 = 248$

Figura 2.4: Ejemplos de multiplicación binaria con reducción.

2.3. Librería de división de secretos

Para aplicar la división de secretos a secuencias de bytes, teniendo en cuenta todos los puntos comentados en secciones anteriores, se ha desarrollado una librería C, a partir del código del programa **ssss**, para dispositivos Windows y Linux. A lo largo de esta sección, se mostrará el funcionamiento de la librería mediante un ejemplo. Este ejemplo es la secuencia de bytes presente en el cuadro 2.1. Esta secuencia de bytes se compone de doscientos (200) elementos.

68	44	c8	32	91	09	d5	96	da	d2	e4	60	09	4d	71	fc	72
5d	3d	f0	3d	64	9b	3b	54	59	9f	6c	42	e7	4c	c2	de	10
aa	6c	13	58	6f	41	d3	4d	e7	42	93	99	b6	d9	7a	b7	aa
bf	7e	4b	f4	a3	4c	eb	27	44	94	e9	51	27	c2	eb	73	27
ec	3c	70	6e	23	4a	eb	f6	d7	d5	4c	57	99	57	86	37	72
dc	9e	05	b1	48	d6	65	68	e6	cd	2a	1a	4c	9f	3d	53	ab
d0	0f	36	6c	48	53	99	2c	86	44	ae	ae	e8	43	b3	25	f1
76	02	a7	62	c5	60	73	4a	a9	b5	20	30	15	47	d6	a0	75
f6	55	9b	8e	bd	19	77	24	ef	71	85	d2	78	42	b8	b6	d0
f4	a6	d1	b1	df	bc	07	31	c5	06	da	50	ec	ef	3b	8e	60
98	dc	99	d9	6f	d7	47	80	d2	23	05	8b	16	ad	15	ee	49
03	fe	38	1d	d6	a5	06	7e	d4	68	93	4c	71				

Cuadro 2.1: Secuencia de bytes a tratar.

La librería desarrollada aplica la división de secretos en bloques de longitud fija. De esta manera, la librería fragmenta el fichero a dividir en bloques, y aplica la división de secretos a cada bloque. Esto permite a la librería fragmentar ficheros de cualquier longitud, a la vez que posibilita la paralelización de las operaciones de la librería.

No obstante, el trabajar con bloques de longitud fija hace que el último bloque de una secuencia de bytes pueda quedar incompleto, es decir, no alcance la longitud del bloque. En estos casos, antes de operar el bloque de bytes, la librería introduce relleno, hasta completar la longitud. Este relleno se realiza introduciendo bytes con valor cero (0x00).

El tamaño de bloque escogido son ciento veintiocho (128) bytes. Esta es la máxima longitud de una secuencia de bytes que el software ssss puede dividir. Esta restricción se debe a los polinomios irreducibles, ya que ssss no puede generar esta clase de polinomios, para campos de Galois muy grandes. El mayor campo de Galois, cuyo polinomio irreducible puede crear es el GF($2^{8 \cdot 128}$).

Se ha optado por la longitud de ciento veintiocho bytes ya que es una longitud no muy larga, lo que asegura que el tamaño de un fichero y sus fragmentos sean muy similares. Además, para procesar ficheros de mayor tamaño, es la longitud de bloque más eficiente de entre las que puede procesar ssss, como se puede observar en la gráfica 2.5.

Esta figura muestra el tiempo necesario para procesar una secuencia de $122,07$ megabytes, para distintos tamaños del bloque. En el eje x viene indicado el ta-

maño del bloque en bytes. En cuanto al eje y, éste se muestra en escala logarítmica, y representa el tiempo en segundos que ha sido necesario para procesar el bloque.

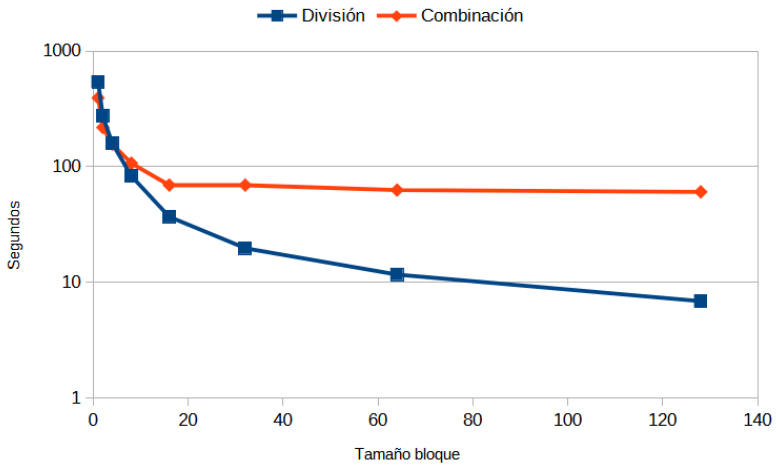

Figura 2.5: Tiempo en procesar $1.28e8$ bytes $(122,07$ MB)

Como bien muestra la gráfica 2.5, para mayor longitud de bloque, menos tiempo se emplea en la división y en la combinación de los bytes. Esto puede deberse a que cuanto más pequeño es el bloque, mayor es el número de operaciones. Por ejemplo, para procesar ciento veintiocho bytes, con bloques de este tamaño solo se tiene que operar una vez. Mientras que para la misma cantidad de bytes, si se emplean bloques de un byte se tiene que operar ciento veintiocho veces; una por cada byte de la secuencia.

En el cuadro 2.2 se muestra la secuencia de bytes original dividida en bloques. Esta división se ha realizado de la misma manera en la que lo haría la librería. En azul se marca el primero bloque que se obtendría de la secuencia de bytes del cuadro 2.1. El segundo bloque que se generaría se encuentra indicado en rojo. Dado que este último bloque no ocupa los ciento veintiocho bytes de un bloque, se introduce relleno. Este relleno se identifica como los bytes pintados de verde.

68	44	c8	32	91	09	d5	96	da	d2	e4	60	09	4d	71	fc	72
5d	3d	f0	3d	64	9b	3b	54	59	9f	6c	42	e7	4c	c2	de	10
aa	6c	13	58	6f	41	d3	4d	e7	42	93	99	b6	d9	7a	b7	aa
bf	7e	4b	f4	a3	4c	eb	27	44	94	e9	51	27	c2	eb	73	27
ec	3c	70	6e	23	4a	eb	f6	d7	d5	4c	57	99	57	86	37	72
dc	9e	05	b1	48	d6	65	68	e6	cd	2a	1a	4c	9f	3d	53	ab
d0	0f	36	6c	48	53	99	2c	86	44	ae	ae	e8	43	b3	25	f1
76	02	a7	62	c5	60	73	4a	a9								

b5	20	30	15	47	d6	a0	75	f6	55	9b	8e	bd	19	77	24	ef
71	85	d2	78	42	b8	b6	d0	f4	a6	d1	b1	df	bc	07	31	c5
06	da	50	ec	ef	3b	8e	60	98	dc	99	d9	6f	d7	47	80	d2
23	05	8b	16	ad	15	ee	49	03	fe	38	1d	d6	a5	06	7e	d4

68	93	4c	71	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00								

Cuadro 2.2: Secuencia de bytes dividida en bloques.

Como se ha mostrado en la sección 2.1 Esquema de Shamir, la división de secretos de realiza sobre valores numéricos. Por este motivo, es necesario guardar cada bloque de bytes como enteros. Dado que cada bloque ocupa ciento veintiocho bytes, es necesario guardar los bloques en una estructura para números grandes.

```
1  #define BASE uint32_t
2  #define NBYTES sizeof(BASE)
3
4  #define MAXDEGREE 1024
5  #define MAXLEN (MAXDEGREE>>3)/NBYTES
6
7  typedef struct {
8      BASE num[MAXLEN+1];
9  } bignum;
10
11 typedef bignum bnm[1];
```

Listing 2.1: Estructura para números grandes

La librería almacena cada bloque en una estructura de números grandes propia, denominada `bnm` (Ver Listing 2.1). Esta estructura se compone de un puntero a una lista de treinta y tres (33) enteros de treinta y dos bits. Estos enteros son enteros sin signo. Se emplean una lista, o array, de treinta y tres elementos, ya que es necesario manejar valores cuya representación emplea más de ciento veintiocho bytes.

En el cuadro 2.3 se muestra como se guardarían los bloques del cuadro 2.2 en la estructura `bnm`. En rojo se marcan los bytes que componen el elemento extra que se añade, y que por tanto no pertenecen a los bloques.

32 c8 44 68	96 d5 09 91	60 e4 d2 da	fc 71 4d 09	f0 3d 5d 72
3b 9b 64 3d	6c 9f 59 54	c2 4c e7 42	6c aa 10 de	41 6f 58 13
42 e7 4d d3	d9 b6 99 93	bf aa b7 7a	a3 f4 4b 7e	44 27 eb 4c
27 51 e9 94	27 73 eb c2	6e 70 3c ec	f6 eb 4a 23	57 4c d5 d7
37 86 57 99	05 9e dc 72	65 d6 48 b1	2a cd e6 68	3d 9f 4c 1a
0f d0 ab 53	53 48 6c 36	44 86 2c 99	43 e8 ae ae	76 f1 25 b3
c5 62 a7 02	a9 4a 73 60	00 00 00 00		

15 30 20 b5	75 a0 d6 47	8e 9b 55 f6	24 77 19 bd	d2 85 71 ef
b6 b8 42 78	d1 a6 f4 d0	07 bc df b1	da 06 c5 31	3b ef ec 50
dc 98 60 8e	d7 6f d9 99	23 d2 80 47	ad 16 8b 05	03 49 ee 15
d6 1d 38 fe	d4 7e 06 a5	71 4c 93 68	00 00 00 00	00 00 00 00

00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00		

Cuadro 2.3: Bloques almacenado en estructuras bnm.

Además, contar con un elemento más, de los necesarios para representar ciento veintiocho bytes, facilita detectar cuando una operación causa desbordamiento. Este elemento extra se guarda con valor cero (0x00), y solo toma otro valor distinto a cero cuando una operación da un resultado fuera del campo de Galois empleado.

Por ejemplo, en el cuadro 2.4 se muestra el resultado de desplazar el primer bloque de los cuadros anteriores un bit a la izquierda. El resultado de este desplazamiento causa desbordamiento, es decir, devuelve un valor que no se puede representar con los ciento veintiocho bytes del bloque. Esto se puede identificar debido a que el elemento extra, marcado en rojo, toma un valor distinto de cero.

65 90 88 d0	2d aa 13 22	c1 c9 a5 b5	f8 e2 9a 12	e0 7a ba e5
77 36 c8 7b	d9 3e b2 a8	84 99 ce 84	d9 54 21 bd	82 de b0 26
85 ce 9b a6	b3 6d 33 26	7f 55 6e f5	47 e8 96 fd	88 4f d6 99
4e a3 d3 28	4e e7 d7 84	dc e0 79 d8	ed d6 94 46	ae 99 ab af
6f 0c af 32	0b 3d b8 e4	cb ac 91 62	55 9b cc d0	7b 3e 98 34
1f a1 56 a6	a6 90 d8 6c	89 0c 59 32	87 d1 5d 5c	ed e2 4b 66
8a c5 4e 04	52 94 e6 c1	00 00 00 01		

Cuadro 2.4: Primer bloque desplazado un bit a la izquierda.

Para conseguir que el resultado se encuentre dentro del campo finito, y por tanto se pueda representar con ciento veintiocho bytes, se tiene que reducir este resultado con el polinomio irreducible. Dicho polinomio irreducible se crea directamente en una estructura `bnm`. El valor del polinomio varía dependiendo de la longitud del bloque con el que se opera. En el cuadro 2.5 se muestra el polinomio irreducible, con el que se operaría el resultado de la tabla 2.4.

00 08 00 43	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 01		

Cuadro 2.5: Polinomio irreducible para $GF(2^{8 \cdot 128})$.

Adicionalmente, para almacenar una secuencia de bytes, es necesario incluir la longitud de la cadena que se quiere almacenar, de manera explícita. Esto se debe a que las secuencias de bytes proceden de ficheros y no de cadenas de texto. Los

ficheros pueden contener entre sus bytes valores que en las cadenas de texto se emplean como caracteres de control.

Un ejemplo de esto es el caracter cero (0x00), el cuál indica el final de una cadena de texto, pero que en un fichero puede ir seguido de otros caracteres. Si se obtuviese el tamaño de la secuencia de bytes a partir de ésta, el segundo bloque del cuadro 2.2 se guardaría parcialmente en la estructura `bnm`.

Una vez se ha guardado el bloque como un número, la librería puede proceder a operar el bloque. Las operaciones que se pueden realizar son la división del bloque y la recuperación de un bloque.

La división de un bloque se realiza empleando el esquema de división de secretos de Shamir (véase sección 2.1 Esquema de Shamir). Para ello, se genera los coeficientes del polinomio con el que obtener los fragmentos. Estos coeficientes se generan como secuencias de bytes aleatorias, tan grandes como el bloque a dividir. En la tabla 2.6 se presentan ejemplos de coeficientes empleados para generar los fragmentos, en un esquema donde se necesitan tres fragmentos para recuperar el bloque original. En rojo se presenta el primer coeficiente (a_1) y en verde el segundo (a_2).

ad 14 88 4e	bc fe 9b 94	45 2b f0 13	db 2f 73 de	8d 86 42 3f
94 06 76 25	b8 03 0d 26	d4 1f 38 f3	64 4d 4e 22	88 29 d4 e2
bd 1e 16 4f	78 96 7d 38	59 4d 33 57	98 6c 01 d4	1f 1a f1 2b
6e d6 de bc	9f c1 83 3a	f4 9e 48 bb	71 5b b7 de	96 c2 29 b7
d4 4f f8 3f	b6 44 31 27	21 9c 1a db	7e 4b 2a 27	48 46 99 e0
9f 31 5a c1	81 1a 31 74	28 90 69 82	a7 18 47 1f	00 4c c4 ba
38 d4 d9 9c	a2 08 3e 82	00 00 00 00		

38 2e 5c 8c	f1 53 41 77	16 65 b5 08	4e da e9 43	6f dc 23 31
25 59 3f 6f	f0 dd 7d aa	b8 4c ec 6e	6c 4a 04 50	2c 0a 7a 86
bc 74 c6 49	5f c2 f9 a8	1f f4 7d c6	d4 13 16 24	69 d9 79 e7
b9 a1 6f 3a	ce 7a 07 21	e5 3e f2 a2	0d fb 58 73	71 d7 92 8e
f3 5f 6c 4b	cd ba 82 a3	64 90 ba 34	75 57 97 fb	d1 91 69 f3
24 53 8b a9	a1 da 3b 9c	f9 c5 c1 7b	7d 30 1b e0	55 17 3a 23
1e 76 63 b1	48 04 f0 0b	00 00 00 00		

Cuadro 2.6: Ejemplos de coeficientes empleados para generar fragmentos.

En la tabla 2.7 se muestran tres fragmentos obtenidos de dividir el primer bloque del cuadro 2.2 a partir del esquema que emplea los coeficientes de la tabla 2.6. En azul se representa el fragmento de índice 1, en rojo el de índice 2 y en verde el de índice 3.

aa	90	f2	a7	72	d3	78	db	c1	97	aa	33	94	d7	84	69	7c
3c	67	12	77	2d	c4	8a	d8	29	41	24	df	33	1f	ae	ac	5a
ad	64	77	f6	4c	e5	d5	9d	8d	43	03	1d	e2	fe	eb	f9	13
f9	8e	5c	8b	ef	80	63	e4	32	12	58	26	f0	d9	6f	c8	76

f5	86	d0	7f	8e	a5	4b	8a	ee	6e	59	b0	ed	c3	96	10	f6
6f	60	7e	5e	e8	da	20	b4	5b	d1	21	09	bc	48	a4	3b	7a
b2	b4	de	66	88	73	60	84	d3	95	51	f2	c0	99	2a	db	aa
23	2f	1d	c0	e3	e9	bd	46	43								
c4	26	58	88	64	3b	65	2a	de	e6	25	b3	b9	0f	44	71	c8
55	41	54	cb	75	f3	86	b1	b5	ec	df	1e	27	41	8b	d9	9d
18	15	ce	1b	15	e1	68	78	08	c9	40	85	91	57	cd	26	e1
72	46	10	60	c3	84	ee	77	dd	05	e8	79	1c	30	f1	18	21
10	67	b7	13	51	44	b1	23	81	cc	96	bd	cb	16	64	52	b2
b4	fc	5f	d5	95	ac	b4	cb	ed	05	03	17	d8	57	eb	76	30
fc	a0	af	e0	14	d6	72	fa	b1	f2	13	4f	18	f9	4b	44	34
22	ff	9a	12	cd	48	ce	49	cd								
06	f2	62	1d	87	e1	c8	67	c5	a3	6b	e0	24	95	b1	e4	c6
34	1b	b6	81	3c	ac	37	3d	c5	32	97	83	f3	12	e7	ab	d7
1f	1d	aa	b5	36	45	6e	a8	62	c8	d0	01	c5	70	5c	68	58
34	b6	07	1f	8f	48	66	b4	ab	83	59	0e	cb	2b	75	a3	70
09	dd	17	02	fc	ab	11	5f	b8	77	83	5a	bf	82	74	75	36
07	02	24	3a	35	a0	f1	17	50	19	08	04	28	80	72	1e	e1
9e	1b	47	ea	d4	f6	8b	52	e4	23	ec	13	30	23	d2	ba	6f
77	d2	20	b0	eb	c1	00	45	27								

Cuadro 2.7: Secuencia de bytes dividida en bloques.

Para recuperar el bloque dividido, se ha optado por emplear la Interpolación Polinómica de Newton [14], en lugar de la Interpolación Polinómica de Lagrange [15]. El motivo por el que se emplea la Interpolación de Newton es porque la implementación de esta, es más rápida que la implementación de la interpolación de Lagrange.

Para la Interpolación de Lagrange, se realizan más productos (operación de multiplicación) que la Interpolación de Newton. De acuerdo a la implementación del producto de dos valores, la multiplicación a nivel de bit en campos finitos (véase sección 2.2 Aritmética modular), requiere más tiempo cuanto más largas son las secuencias de bytes a operar. Esta es la causa, de que el multiplicar dos secuencias de veintiocho bytes sea una operación bastante lenta, y por tanto, la Interpolación de Newton permita recuperar el secreto más rápidamente que la Interpolación de Lagrange, en este caso.

Tras la operación de interpolación, se devuelve el bloque interpolado. Este bloque se presupone que es el bloque original. No obstante, la librería no cuenta con ningún método mediante el que comprobar si el bloque recuperado coincide con el bloque original.

[14](2023) Wikipedia - Interpolación polinómica de Newton. [Online]. Available: `https://en.wikipedia.org/wiki/Newton_polynomial`

[15](2023) Wikipedia - Interpolación Polinómica de Lagrange. [Online]. Available: `https://en.wikipedia.org/wiki/Lagrange_polynomial`

Para conseguir todo esto, se han realizado modificaciones al código fuente del software ssss. El principal cambio realizado ha sido modificar el método empleado para obtener los bytes a operar, así como la forma en la que se devuelven los bytes resultantes de dichas operaciones. Mediante este cambio, en lugar de emplear la entrada y salida estándar, el código recibe los bytes a procesar como un parámetro, y almacena los resultantes en otro parámetro diferente.

El resto de cambios se han centrado en acelerar las distintas operaciones aritméticas, sobre todo la multiplicación, y en acelerar la recuperación de los bloques originales.

Por último, la librería desarrollada paraleliza tanto la división de secretos como la recuperación de estos empleando hilos del sistema operativo. Si la librería se ejecuta en un equipo con un sistema operativo Windows, estos hilos serán Threads de Windows [16], mientras que si se ejecuta en una máquina Linux, se emplearán pthreads [17].

En concreto, la librería genera un máximo de dieciséis (16) hilos, que se reparten el fichero a fragmentar o recuperar en bloques de un megabyte (1 MB). Si se empleasen bloques de mayor tamaño, la cantidad de memoria necesaria para operar puede superar la cantidad de memoria que se puede reservar. Por otro lado, si el tamaño de bloque es inferior a un megabyte, la cantidad de operaciones de lectura y escritura en disco causan que el funcionamiento de la librería sea demasiado lento.

La repartición del fichero entre los hilos se realiza a partir de un índice del hilo. Al primer hilo que se crea se le asigna el índice cero (0), al segundo hilo el índice uno (1) y así hasta el hilo dieciséis. Con los índices asignados, cada uno de los hilos sigue la fórmula $bloque = indice + 16 \cdot x$ para calcular los bloques que procesar. De esta forma, el primer hilo se encargaría del bloque cero (bytes $[0, \ldots, 1 \cdot 2^{20}]$), el bloque dieciséis (bytes $[16 \cdot 2^{20}, \ldots, 17 \cdot 2^{20}]$), etc. hasta que el bloque a procesar supere el tamaño en megabytes del fichero.

En el caso de que el fichero a tratar sea inferior a los dieciséis megabytes (16 MB), no se inician todos los hilos, sino que se generan tantos hilos como bloques de un megabyte sean necesarios para fragmentar o recuperar el fichero. Por ejemplo, si el fichero a procesar ocupa $9,2$ MB, la librería genera diez (10) hilos para el procesamiento del fichero.

El funcionamiento de los hilos consiste en la lectura y escritura en bloques de un megabyte, de los ficheros correspondientes para aplicar la división o recuperación del fichero en bloques de ciento veintiocho bytes. De manera más concreta, cada hilo lee un megabyte de los ficheros correspondientes, opera sobre los megabytes leídos en bloques de ciento veintiocho bytes, y escribe los megabytes resultantes en los ficheros correspondientes.

[16](2023) Threads de Windows. [Online]. Available: `https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads`

[17](2023) Wikipedia - Threads de POSIX (pthreads). [Online]. Available: `https://en.wikipedia.org/wiki/Pthreads`

Las funciones que presta la librería al usuario son `division`, que aplica la división de secretos, y `combina`, con la que se recupera el fichero fragmentado. La función `division` recibe como parámetros el threshold o cantidad mínima de fragmentos necesarios para recuperar el fichero (k), la cantidad de fragmentos a generar (n), el tamaño del fichero a fragmentar (tam), la ruta del fichero a fragmentar (fichero) y una lista con las rutas de los fragmentos a generar (frags).

En cuanto a la función `combina`, esta toma como parámetros el threshold (k), el tamaño del fichero a recuperar (tam), la ruta en la que guardar el fichero recuperado (fichero) y dos listas. La primera lista (frags) contiene las rutas de los fragmentos con los que recuperar el fichero. La segunda lista (indices) contiene los índices de los fragmentos de la lista *frags*, de tal manera que el primer índice de la lista *indices* es el correspondiente al primer fragmento de *frags*, el segundo índice se corresponde con el segundo fragmento y así sucesivamente.

Las cabeceras de estas dos funciones están presentes en el Listing 2.2

```
1 int division(int k, int n, int tam, char *fichero, char *frags[]);
2 int combina(int k, int tam, char *fichero, char *frags[], int indices[]);
```

Listing 2.2: Cabeceras de las funciones de la librería

Capítulo 3

Aplicación Desarrollada

En este capítulo se explica la aplicación desarrollada como objetivo de este proyecto. El software desarrollado consiste en una aplicación de escritorio para dispositivos Windows y Linux, desarrollada en lenguaje Python. Estos sistemas operativos son los únicos para los que la librería desarrollada está preparada para una correcta ejecución. En otras palabras, son los únicos sistemas operativos para los que se ha compilado y generado la librería.

La aplicación desarrollada es un agregador de nubes, por tanto combina el espacio de distintas cuentas de almacenamiento en la nube. Mediante esta aplicación, el usuario final puede almacenar sus datos en todas las cuentas que vincule, como si fuese una única cuenta de almacenamiento en la nube.

De esta manera, la aplicación actúa como una intermediaria entre el usuario final y los servicios en la nube, en la gestión de los datos de los usuarios en forma de ficheros. Estos ficheros son unidades de almacenamiento lógicas que agrupan conjuntos de datos relacionados entre sí.

Como intermediaria, la aplicación protege los ficheros de los usuarios cifrándolos y aplicando la división de secretos, con el objetivo de asegurar la confidencilidad de los datos, así como aumentar la disponibilidad que el usuario puede tener de estos. A su vez, esta aplicación comprueba durante la recuperación de un fichero la integridad de sus datos, para asegurar que el contenido de este no ha sido modificado.

Toda la información que este agregador necesita guardar de manera persistente, se recoge en una base de datos. Para evitar que se pueda acceder a los contenidos de esta base de datos, la base de datos se guarda cifrada en el sistema de ficheros. Mientras tanto, la aplicación guarda, durante su ejecución, los contenidos descifrados en una base de datos en memoria RAM [1]. Para gestionar la base de datos, la aplicación hace uso el software SQLite [2].

Por último, el agregador cuenta con una interfaz gráfica. Esta interfaz gráfica

[1](2023) Bases de datos en memoria. [Online]. Available: https://www.sqlite.org/inmemorydb.html

[2](2023) SQLite. [Online]. Available: https://www.sqlite.org/index.html

está creada empleando el módulo Python Kivy [3]. Kivy permite la creación de interfaces gráficas multiplataforma.

3.1. Almacenamiento en la nube

Como se ha comentado, esta aplicación actúa como intermediaria entre un usuario final y múltiples cuentas de almacenamiento en la nube. Para conseguir esto, se han desarrollado varios módulos en los que cada uno se encarga de establecer las conexiones, y realizar el intercambio de datos, con un proveedor de almacenamiento en la nube concreto.

Los servicios de almacenamiento en la nube compatibles con la aplicación, aquellos que cuentan con un módulo son: Mega [4], PCloud [5], Google Drive [6], OneDrive [7] y Dropbox [8]. La API empleada para cada uno de estos servicios, se encuentra presente en el Cuadro 3.1. Además de la API, en el cuadro se han clasificado los servicios en la columna `tipo de conexión`, de acuerdo al método empleado por cada API para establecer la conexión con una cuenta.

Servicio	**Proveedor**	**API**	**Tipo de conexión**
Mega	Mega	mega.py [9]	Correo + contraseña
PCloud	PCloud	pycloud [10]	Correo + contraseña
Google Drive	Google	google-api-python [11]	OAuth
OneDrive	Microsoft	microsoft graph [12]	OAuth
Dropbox	Dropbox	dropbox-sdk-python [13]	OAuth

Cuadro 3.1: Nubes con soporte en la aplicación

El primer grupo de esta clasificación, son las APIs que emplean el correo vinculado a la cuenta, y la contraseña de ésta. Estas APIs coinciden con los servicios en la nube que emplean el cifrado por parte del cliente. Es por ello que supuestamente, la API toma la contraseña, y la emplea únicamente para obtener las claves necesarias con las que conectarse a la cuenta, y con las que cifrar y descifrar los datos que se suben.

El otro grupo de la clasificación, son aquellas APIs que emplean la autenticación OAuth [14]. Este método de autenticación redirige al usuario al proveedor del almacenamiento en la nube, para ser autenticado. Si la autenticación con el proveedor es correcta, la aplicación recibe un token de autenticación. Estos tokens de autenticación se pueden emplear para establecer conexiones con la cuenta, sin la necesidad de que el usuario se tenga que autenticar de nuevo. Estos tokens son elementos JSON [15].

[3](2023) Kivy. [Online]. Available: `https://kivy.org/`
[4](2023) MEGA. [Online]. Available: `mega.nz/`
[5](2023) PCloud. [Online]. Available: `https://www.pcloud.com/es/`
[6](2023) Google Drive. [Online]. Available: `https://drive.google.com/`
[7](2023) OneDrive. [Online]. Available: `https://onedrive.live.com/about/es-es/`
[8](2023) Dropbox. [Online]. Available: `dropbox.com/`
[14](2023) OAuth 2.0. [Online]. Available: `https://oauth.net/2/`
[15](2023) Wikipedia - JSON. [Online]. Available: `https://es.wikipedia.org/wiki/JSON`

Además, las APIs que emplean autenticación OAuth, requieren que la aplicación desarrollada se registre en el proveedor de la nube.

Todos los módulos creados comparten las mismas funcionalidades dadas por las funciones o métodos presentes en el Listing 3.1.

```
1 __init__(token) -> Conexion
2 subir(fichero) -> None
3 descargar(fichero,destino) -> None
4 borrar(fichero) -> None
5 getCredendiales() -> str
6 espacio() -> int
7 getUser() -> str
```

Listing 3.1: Funciones de los módulos de conexión a nubes de almacenamiento

Al crear una nueva instancia de un módulo se llama de manera automática a la función "`__init__`", la cuál genera una nueva conexión con una cuenta del servicio de almacenamiento en la nube correspondiente al módulo.

Para la creación de una nueva conexión se puede pasar como parámetro un objeto JSON [16], el cuál se emplea como token. Este parámetro varía dependiendo de la nube de almacenamiento con la que se esté estableciendo la conexión. En el caso de las nubes cuya conexión se realiza mandando el correo vinculado a la cuenta y la contraseña de la cuenta, este objeto JSON es obligatorio y tiene que contener el correo y la contraseña de la cuenta a la que se quiere conectar. Estos dos valores se mandan en formato JSON, para facilitar la abstracción de los métodos de los módulos.

Si la conexión se establece mediante el protocolo OAuth, este JSON es opcional. En caso de emplearse, tiene que ser un token de acceso a la cuenta con la que se quiere establecer la conexión.

Los métodos "`subir`", "`descargar`" y "`borrar`" se encargan de guardar, descargar y eliminar un fichero de la cuenta con la que se ha establecido la conexión. Este fichero se indica mediante el parámetro *fichero*, el cuál contiene la ruta absoluta local en el método "`subir`", y el nombre del fichero en los métodos "`descargar`" y "`borrar`". El método "`descargar`" recibe un segundo parámetro (*destino*), el cuál toma el valor de la ubicación del sistema de ficheros, donde se quiere guardar el fichero a descargar.

La función "`getCredenciales`" se emplea para obtener un objeto JSON con el que poder conectarse a la cuenta en futuras conexiones. Para las conexiones establecidas mediante un correo electrónico y la contraseña de la cuenta, esta función devuelve el mismo objeto JSON que se empleó para establecer la conexión. En cuanto a las conexiones creadas mediante el protocolo OAuth, el JSON que devuelve esta función es un token de acceso con el que otra instancia del mismo módulo puede conectarse a la misma cuenta.

[16] T. Bray, "The javascript object notation (json) data interchange format,"Internet Engineering Task Force (IETF), RFC 8259, dec 2017. [Online].Available: `https://www.rfc-editor.org/rfc/rfc8259`

Las funciones "`espacio`" y '`getUser`" son funciones que permiten obtener información acerca de la cuenta conectada. La primera de éstas funciones ("`espacio`") devuelve el espacio libre que queda en la cuenta, es decir la cuota de espacio asignado a la cuenta, que el usuario todavía no ha empleado. La función "`getUser`" se emplea para obtener el usuario al que pertenece la cuenta. Dado que en los servicios de almacenamiento, un usuario solo puede tener una cuenta, la principal funcionalidad que se da a "`getUser`" es comprobar si ya se ha vinculado una cuenta de un servicio de almacenamiento con la aplicación. Cabe destacar que un usuario puede tener varias cuentas en distintos servicios de almacenamiento, por lo que una cuenta de almacenamiento se puede identificar por el proveedor del servicio de almacenamiento y el usario al que pertenece la cuenta.

3.2. Funcionamiento de la aplicación

El agregador de almacenamientos en la nube consiste en una aplicación que gestiona los ficheros de un usuario entre varios almacenamientos en la nube. De manera más concreta, la aplicación desarrollada permite al usuario subir ficheros al espacio en la nube, descargar ficheros de este espacio y eliminarlos.

3.2.1. Subida de ficheros

La primera funcionalidad sobre ficheros, que presenta el agregador de nubes de almacenamiento, es la subida de un fichero a la nube. Mediante esta función, el usuario da un fichero a la aplicación, para que esta lo gestione en la nube. En este punto se detalla el proceso que sigue la aplicación para proteger y guardar el fichero en remoto.

La subida de un fichero comienza con el usuario indicando el fichero que quiere guardar en la nube. En la figura 3.1 se muestra la pantalla con la que seleccionar un fichero. En este caso, se escoge subir un fichero de doscientos bytes, el cuál se empleará como ejemplo a lo largo de esta sección. El contenido de este fichero se encuentra representado en el cuadro 3.2.

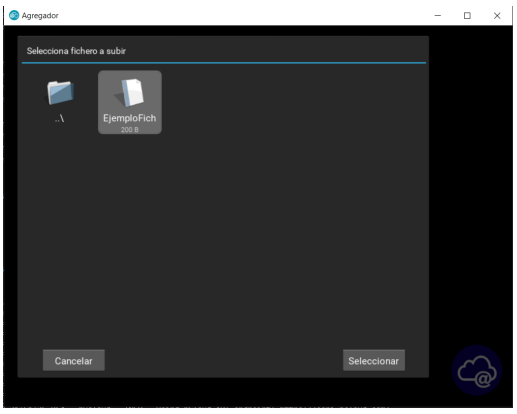

Figura 3.1: Pantalla selección fichero

68	44	c8	32	91	09	d5	96	da	d2	e4	60	09	4d	71	fc	72
5d	3d	f0	3d	64	9b	3b	54	59	9f	6c	42	e7	4c	c2	de	10
aa	6c	13	58	6f	41	d3	4d	e7	42	93	99	b6	d9	7a	b7	aa
bf	7e	4b	f4	a3	4c	eb	27	44	94	e9	51	27	c2	eb	73	27
ec	3c	70	6e	23	4a	eb	f6	d7	d5	4c	57	99	57	86	37	72
dc	9e	05	b1	48	d6	65	68	e6	cd	2a	1a	4c	9f	3d	53	ab
d0	0f	36	6c	48	53	99	2c	86	44	ae	ae	e8	43	b3	25	f1
76	02	a7	62	c5	60	73	4a	a9	b5	20	30	15	47	d6	a0	75
f6	55	9b	8e	bd	19	77	24	ef	71	85	d2	78	42	b8	b6	d0
f4	a6	d1	b1	df	bc	07	31	c5	06	da	50	ec	ef	3b	8e	60
98	dc	99	d9	6f	d7	47	80	d2	23	05	8b	16	ad	15	ee	49
03	fe	38	1d	d6	a5	06	7e	d4	68	93	4c	71				

Cuadro 3.2: Secuencia de bytes del fichero a subir.

Además, se solicita al usuario que indique el número de fragmentos a generar. Junto a esta, también se pide que indique la cantidad de fragmentos generados, son neccesarios para recuperar el fichero. La pantalla en la que el usuario introduce estos dos valores, es la que se encuentra en la figura 3.2. Los valores que se introudcen en esta pantalla son los que se emplean para el ejemplo de esta sección.

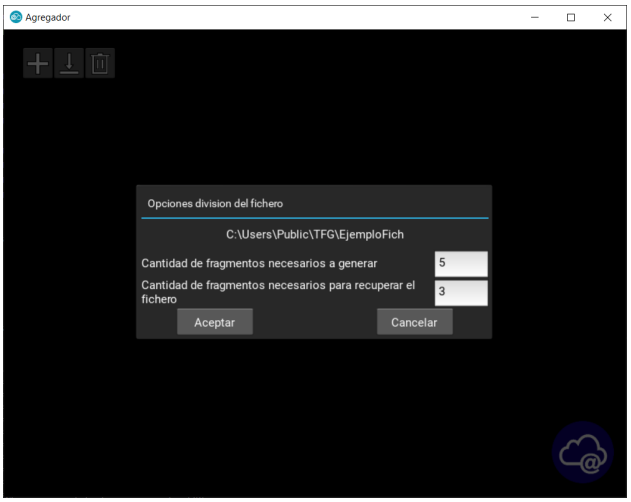

Figura 3.2: Pantalla para configurar la división de secretos

Con toda esta información, la aplicación comienza la subida comprobando que se pueda almacenar el fichero en la nube. Para ello, el agregador procede a escoger las cuentas en las que subir los fragmentos. Para seleccionar las cuentas a emplear, el agregador recupera de la base de datos todas las cuentas que se han vinculado. Esta lista de cuentas contiene para cada una el servicio en el que se encuentra la cuenta, y el token con el que establecer la conexión.

Cada uno de los elementos de esta lista se selecciona de manera aleatoria, para evitar emplear siempre las mismas nubes. Esta selección se realiza hasta haber comprobado todas las cuentas, o hasta que se ha establecido conexión con

tantas cuentas como fragmentos se quieren generar. En este segundo caso, cada una de las cuentas con las que se ha establecido conexión tiene que tener espacio suficiente como para guardar un fragmento.

Durante la selección de estas cuentas, se intenta establecer la conexión. Si la conexión a la cuenta falla, o el espacio libre en ésta es inferior al tamaño de un fragmento, se descarta la cuenta para subir el fichero y se pasa a la siguiente. En cualquier otro caso se guarda la conexión en una lista de conexiones para su posterior uso.

Al finalizar de seleccionar las cuentas, se comprueba que el fichero puede guardarse de manera segura y correcta. Primero se comprueba que todas las cuentas tienen que almacenar menos fragmentos de los necesarios para recuperar el fichero. Es decir, se comprueba que el secreto no se puede recuperar a partir de una única cuenta.

Por otro lado, si no se tienen tantas conexiones como fragmentos, se realiza una segunda comprobación del espacio. En este proceso, la aplicación calcula la cantidad de fragmentos que se pueden almacenar en las cuentas disponibles. Si algunas de las dos revisiones falla, no se puede almacenar el fichero de manera segura, y por tanto se suspende la subida del fichero en la nube.

Tras establecer las conexiones, el agregador continúa leyendo los contenidos del fichero con dos objetivos. El primer objetivo de esta lectura es obtener un *hash* del fichero, para lo que se emplea la función hash SHA256 [17], que devuelve una secuencia de doscientos cincuenta y seis bits.

Un *hash* es el resultado de aplicar una función *hash* [18]. Gracias a las propiedades de este tipo de funciones, la aplicación puede almacenar el *hash* como un valor de longitud inferior al tamaño del fichero, con el que poder comprobar la integridad del mismo cuando se recupere.

El otro objetivo de leer el fichero antes de subirlo a la nube es proteger sus contenidos cifrándolo. Para proteger el fichero se emplea el algoritmo AES [19], en modo CBC, empleando claves de doscientos cincuenta y seis bits. Este modo requiere que el fichero se pueda dividir en bloques de dieciséis bytes. Si el último bloque ocupa menos de dieciséis bytes, se introducen tantos bytes con valor cero (0x00), hasta conseguir que el bloque ocupe dieciséis bytes.

Se emplea AES en modo CBC, porque es el modo de cifrado en bloque que se considera más seguro. Frente al cifrado en bloques, otra solución son los cifrados autenticados, los cuales además de cifrar y descifrar los datos que reciben, comprueban la integridad de estos. No obstante, se ha optado por emplear un modo de cifrado en bloques, puesto que la integridad del fichero ya la comprueba el agregador de manera independiente al cifrado.

[17] I. T. Laboratory, "Secure hash standard (shs)," FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, no. 180-4, aug 2015.

[18] (2023) Wikipedia - Funciones Hash. [Online]. Available: `https://en.wikipedia.org/wiki/Hash_function`

[19] .Daemen and V. Rijmen, The design of Rijndael: AES — the Advanced Encryption Standard. Springer-Verlag, 2002.

Hay que remarcar que existen modos de cifrado en bloque, como el XTS, cuyo uso también se recomienda. Sin embargo, el uso de éstos últimos está orientado al cifrado de bloques de memoria en disco.

La copia cifrada del fichero se guarda en el sistema de ficheros de manera temporal para poder aplicarle la división de secretos. Al cifrar el fichero, la aplicación se asegura que aún recuperándose, solo el usuario propietario de éste es capaz de obtener el fichero en claro.

Tanto las claves de cifrado de un fichero, como el valor inicial necesario para cifrarlo, se obtienen de manera aleatoria. Estos dos valores, junto al hash del fichero, se guardan en la base de datos.

Siguiendo con el ejemplo establecido al principio de esta sección, el fichero se cifraría y se obtendría su hash. En el cuadro 3.3 se indican la clave de cifra en rojo y el valor inicial en verde. En el cuadro 3.4 y en azul, se indica el hash del fichero. Por último, en el cuadro 3.5 presenta en naranja el resultado de cifrar el fichero.

30	44	a4	ee	d1	f2	4f	d2	8f	46	2e	67	93	bd	1a	e5	02
20	f7	92	0d	bb	19	de	e9	99	0f	d4	16	2b	fa	39	70	b2
c8	2a	0e	3c	db	c3	a9	e3	09	fd	35	f2	9b	3e	2d	58	82
6a	15	f7	32	51	54	bb	3d	f5	90	dc	3d	24				
d8	01	21	c4	8c	07	4b	1e	d5	cd	01	90	70	a0	4a	6b	

Cuadro 3.3: Clave de cifrado del fichero y valor incial

19	2e	9b	58	23	c5	58	13	66	19	e7	3c	3e	b7	5d	75	ac
8d	86	41	0a	17	80	99	f4	4a	6c	0a	c0	e4	11	d4		

Cuadro 3.4: Hash del fichero

b2	19	68	42	c6	ce	0e	9d	87	be	8e	8e	d9	8c	5f	5f	80
d2	41	ba	c9	6f	94	10	62	22	d4	61	65	81	74	42	e0	cb
ee	05	27	32	c6	4c	6f	21	1b	73	74	9f	36	f2	d9	35	f4
5c	12	80	f6	91	b3	ab	2c	ba	f0	6c	ce	58	bb	bc	4e	a6
0c	03	62	1e	c2	2b	ff	c3	ce	e5	cf	5f	65	d5	12	ab	da
a7	94	9e	91	f0	72	53	e1	8c	fb	8f	e9	dd	dc	97	35	6a
1e	aa	8e	18	0b	55	6f	22	d9	e1	01	a5	77	0a	2e	7d	74
72	df	d6	b7	92	0b	74	ac	22	66	cb	0a	c1	4f	91	d4	d7
cb	66	a4	6c	8c	68	39	de	3b	d9	21	af	46	e0	21	8e	7d
27	b9	ac	99	31	9f	22	a6	52	4b	6f	4d	91	41	12	2f	ae
37	a7	27	ac	8e	7b	74	a2	92	67	3e	4c	be	da	ed	99	78
7c	8f	49	13	d6	9a	e4	73	94	66	d6	6c	1d				

Cuadro 3.5: Fichero cifrado

El siguiente paso que realiza la aplicación desarrollada es aplicar la división de secretos al fichero cifrado. Entre los parámetros necesarios para la división son los fragmentos a generar. Para evitar que estos fragmentos proporcionen información alguna del fichero, a parte de aplicar la división de secretos, el nombre de los fragmentos se genera empleando la función hash SHA256.

Para generar el nombre del fragmento, se pasa a la función hash como valor de entrada una cadena en la que se incluye el nombre del fichero con el que se genera el fragmento, y el índice del fragmento. Con esto se consigue que el nombre del fichero dependa de información única a cada fragmento, pero no proporcione esta información.

En el listing 3.2 se muestra como se genera el nombre de un fragmento. En el cuadro 3.6 se presenta el nombre generado para los cinco fragmentos del ejemplo de la sección.

```
1     nombreFragmento = SHA256("Fragmento " + nombreFichero + indiceFragmento)
2         nombreFichero = nombre fichero a fragmentar
3         indiceFragmento = indice fragmento
```

Listing 3.2: Generación del nombre de un fragmento

Fragmento	**Nombre del fragmento**
Fragmento 1	e682e2ddb542979775c7a9e31d4169c7f7542fddae080704be729670146a1872
Fragmento 2	a9aa6f2e0603ac10d9bff36fd4d551ca9bb800cde9e364c03065a879e328fdbb
Fragmento 3	9867ac9786bb5dfa1db6019863a6983679c151262f2938549f85bec4abae6565
Fragmento 4	68afa5b4f0eef6c9e5b1951f5140fe66079900d59191b2f4dfe8f55b981665bf
Fragmento 5	ec621e0ce1f01bab63a7243c8a16b990f4bc1aa5b9277bd0ca84b5b2c57fb742

Cuadro 3.6: Nombres de los fragmentos

Con el nombre de todos los fragmentos calculados, se llama a la librería de división de secretos (véase sección 2.3 Librería de división de secretos) para que fragmente el fichero cifrado y cree los fragmentos con los nombres dados. La librería lee del disco el fichero cifrado, y escribe directamente en disco los fragmentos. Los fragmentos se guardan en el sistema de ficheros, en la misma ubicación temporal que el fichero cifrado. En el cuadro 3.7 se presenta el fragmento de índice 1, generado en el ejemplo de la sección.

85	ae	b7	59	e1	22	92	fd	c3	8f	50	ec	26	3a	08	8b	4b
8d	5f	1a	70	a5	68	3a	8a	09	8f	9a	cc	93	1e	86	40	04
15	44	1e	f6	b9	e3	de	08	a7	ab	f3	35	56	78	b0	0c	cf
22	34	23	24	c1	4f	63	48	43	d8	71	df	30	a3	8d	f8	7e
08	14	ef	52	e8	fd	40	b8	44	aa	8d	80	c9	19	e6	84	55
65	82	f7	4d	8e	89	2d	51	cf	45	4f	a3	c3	db	49	c1	47
22	06	89	24	04	b4	a4	34	20	29	09	d5	d2	a1	f9	3c	16
ae	fe	9c	47	60	71	fe	b0	d3								

Cuadro 3.7: Fragmento 1 del fichero a subir.

Una vez se tienen generados todos los fragmentos, se calcula el valor *hash* de cada uno de ellos, empleando SHA256. Estos hashes se guardan en la base de datos, para sus respectivos ficheros. El hash del fragmento presente en el cuadro 3.7 es el representado en el cuadro 3.8.

db	6f	1b	76	20	15	81	f3	62	97	5f	81	a8	37	9b	75	9d
55	96	1a	a1	b9	36	d5	c7	cb	26	cd	22	a9	3d	78		

Cuadro 3.8: Hash del fragmento 1 del fichero a subir.

Con los fragmentos y los respectivos hashes calculados, la copia cifrada del fichero se borra del sistema de ficheros. Con esta acción se pretende evitar que se ocupe mucha memoria del disco, sobre todo a largo plazo y habiendo subido ficheros grandes.

Tras dividir el fichero cifrado, el agregador procede a subir cada uno de los fragmentos a una de las cuentas disponibles. Para ello, se guarda el primer fragmento en la primera cuenta con la que se conectó, el segundo fragmento en la segunda cuenta, etc. En el supuesto de que se hayan establecido menos conexiones que fragmentos se han generado, la aplicación sube más de un fragmento en algunas nubes. Por ejemplo, si se quieren guardar cinco fragmentos, pero solo se han podido establecer tres conexiones, la primera cuenta almacena los fragmentos uno y cuatro, la segunda almacena los fragmentos dos y cinco y la tercera el fragmento tres.

Si no se pueden subir todos los fragmentos a la nube, la aplicación interrumpe la operación de subida, elimina los fragmentos que se hayan subido y notifica del fallo al usuario. Por el contrario, si la operación finaliza sin que se genere ningún problema, el agregador guarda los datos fichero en la base de datos. También almacena en la base de datos la información de los fragmentos de dicho fichero, como son el índice y la cuenta a la que se ha subido.

Tanto si se produce algún fallo al final, como si la subida finaliza correctamente, la aplicación elimina del almacenamiento local los fragmentos. Los fragmentos se eliminan para evitar, sobre todo si se suben ficheros grandes, que a largo plazo la aplicación pueda ocupar mucho espacio del disco. Debido a esto, tras finalizar la subida de un fichero, en el dispositivo del usuario solamente permanece el fichero original que se quiso subir.

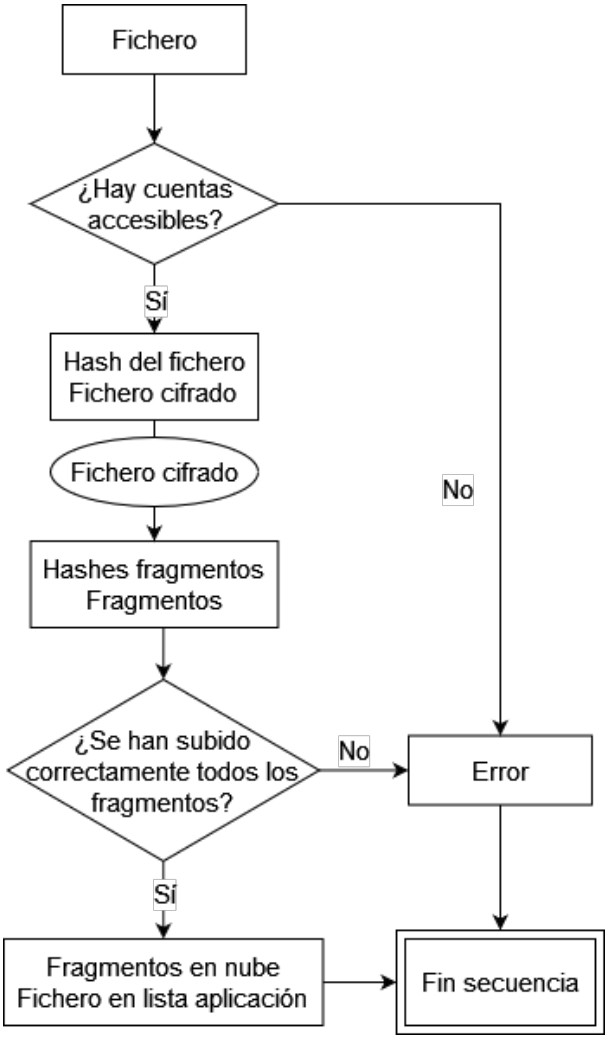

Figura 3.3: Diagrama lógico seguido para la subida de un fichero

3.2.2. Descarga de ficheros

Otra de las funcionalidades que presenta el agregador de almacenamiento en la nube es la recuperación y descarga de ficheros gestionados por el agregador. Esta función permite al usuario recuperar uno de sus ficheros. El fichero descargado se guarda en la carpeta de "Descargas" del dispositivo en el que se ejecuta la aplicación. Se ha decidido emplear esta carpeta, ya que es la carpeta en la que la mayoría de aplicaciones guardan por defecto datos descargados.

Esta operación comienza con el usuario indicando a la aplicación el fichero que quiere descargar. El usuario solo puede solicitar la descarga de ficheros que fueron subidos por el agregador. Esta selección se presenta en la figura 3.4. En este caso, el fichero que se selecciona es el mismo que se empleó como ejemplo en la sección 3.2.1.

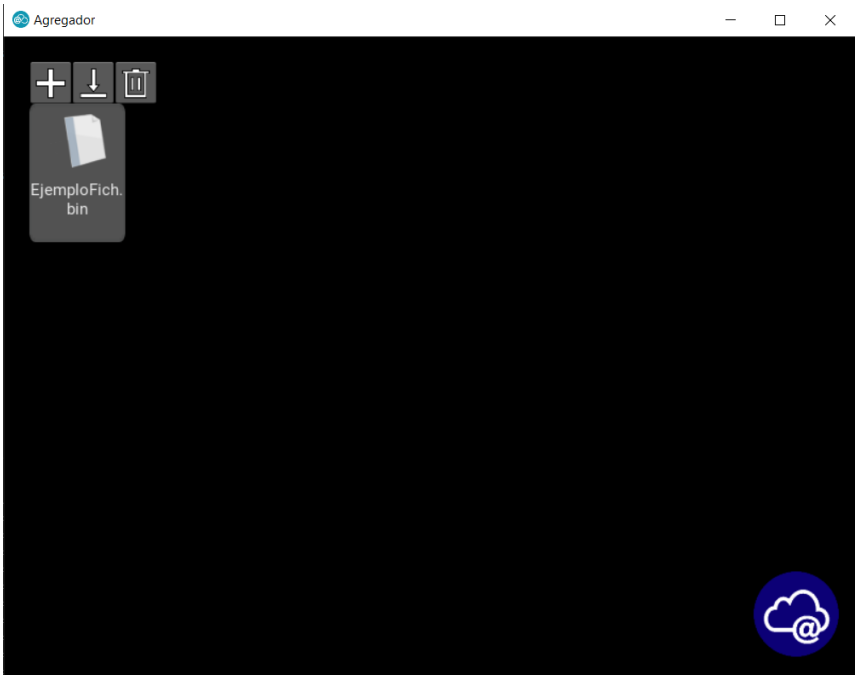

Figura 3.4: Pantalla en la que se selecciona el fichero a descargar

Con el fichero seleccionado, la aplicación comienza recuperando de la base de datos la información acerca del fichero. Entre estos datos se encuentran el hash de este fichero, la cantidad de fragmentos necesaria para recuperarlo y la clave y valor inicial con los que fue cifrado.

Además, el agregador también recoge de la base de datos la información acerca de los fragmentos con los que poder recuperar el fichero. De cada uno de estos fragmentos recupera el índice, el hash del fragmento y la cuenta en la que se guardó durante la subida. Conociendo las cuentas en las que se encuentran los fragmentos, la aplicación continua la recuperación y descarga, obteniendo de la nube los fragmentos.

Para descargar un fragmento, se intenta establecer conexión con el espacio en la nube donde se subió previamente. Si la conexión falla, la aplicación descarta el fragmento para esa descarga y pasa a intentar obtener el siguiente fragmento. Por el contrario, si se consigue conectar correctamente al espacio de almacenamiento, el fragmento se descarga en el sistema de ficheros local.

Con una copia del fragmento en el dispositivo, la aplicación calcula el *hash* de la copia y se compara con el guardado, en la base de datos, de ese fragmento antes de que se guardara en remoto. Esta comprobación es necesaria, ya que en el caso de que ambos valores difieran, el fragmento recuperado no es el mismo que el que se subió a la nube.

La discrepancia entre ambos valores hash puede deberse a que el servicio en la nube no ha devuelto el fragmento que se solicitó. Otra posible causa de que ambos valores difieran es que se haya producido un fallo durante la subida o

la descarga del fragmento. Dado que el hash de un fragmento se calcula justo antes de subir el fragmento a la nube, estas son las principales causas de que ambos valores difieran.

Dado que si ambos valores son distintos el fragmento descargado no es el mismo que se subió a la nube, se descarta el fragmento y se elimina del dispositivo. Mientras que si coinciden, se acepta que el fragmento descargado es el mismo que el que se subió, por lo que se guarda el fragmento en una ubicación temporal del disco. A su vez, se guarda también el índice de este fragmento, para su posterior uso en la recuperación de secretos. En ambos casos, se pasa al siguiente fragmento.

La descarga de fragmentos se realiza hasta que se ha descargado la cantidad mínima de fragmentos, necesaria para recuperar el fichero, o hasta que se ha intentado recuperar todos los fragmentos de la nube. Al finalizar este proceso, si se tienen menos fragmentos de los necesarios para recuperar el fichero, la aplicación da error por la imposibilidad de obtener el fichero original. Además, elimina del almacenamiento local los fragmentos que consiguió recuperar, volviendo así al estado anterior a comenzar la descarga.

Si se tiene la cantidad de fragmentos con la que poder recuperar el fichero, la aplicación continúa llamando a la librería de división de secretos (véase sección 2.3 Librería de división de secretos). Mediante esta llamada, la aplicación recupera el fichero cifrado.

Tras recuperar el fichero cifrado, los fragmentos dejan de ser necesarios en el espacio local. En este momento se entiende que los fragmentos siempre van a recuperar el fichero devuelto. Por este motivo, se eliminan del dispositivo.

El agregador continúa descifrando el fichero recuperado y calculando el hash del resultante. Una vez la aplicación termina de descifrar el fichero, cuenta con una copia del posible fichero recuperado y el hash de este fichero.

El hash del fichero descifrado se compara con el hash que se calculó del fichero original, antes de que se cifrara y se subiera a la nube. En el caso de que el fichero descifrado fuese el fichero original que se ha querido descargar, ambos valores hash coincidirán. Por el contrario, si ambos valores no son el mismo, se ha producido un error o bien en el proceso de cifrado y de división de secretos, o bien en la recuperación y descarga del fichero.

En el caso de que ambos valores coincidan, se entiende que se ha recuperado el fichero original y la operación de recuperación y descarga finaliza. Por el contrario, si los dos valores son diferentes, el fichero obtenido es eliminado y se notifica del error al usuario. En ambos casos, el fichero cifrado se elimina del almacenamiento local.

Error	**Resultado**	**Posibles Causas**
No se puede acceder a una cuenta	Se ignora la cuenta en la descarga del fichero	No se está conectado a la red. El servicio no está disponible
No se puede descargar el fragmento	Se ignora el fragmento en la descarga del fichero	El fichero no existe en la cuenta. Se ha perdido la conexión con el servicio
El hash del fragmento descargado no coincide con el del fragmento que se subió	El fragmento no se toma como válido	El fragmento ha sido modificado en la subida o descarga del fichero. El fragmento ha sido modificado sin intervención del agregador.
No se pueden descargar suficientes fragmentos como para recuperar el fichero	Se interrumpe la descarga y recuperación	
El hash del fichero obtenido no coincide con el del fichero que se subió	Se da por errónea la recuperación del fichero	Se ha producido un fallo en el cifrado/descifrado del fichero. Se ha producido un error en la división de secretos. Se ha producido un error en la recuperación del fichero.

Cuadro 3.9: Errores que se pueden producir de la descarga y recuperación de un fichero

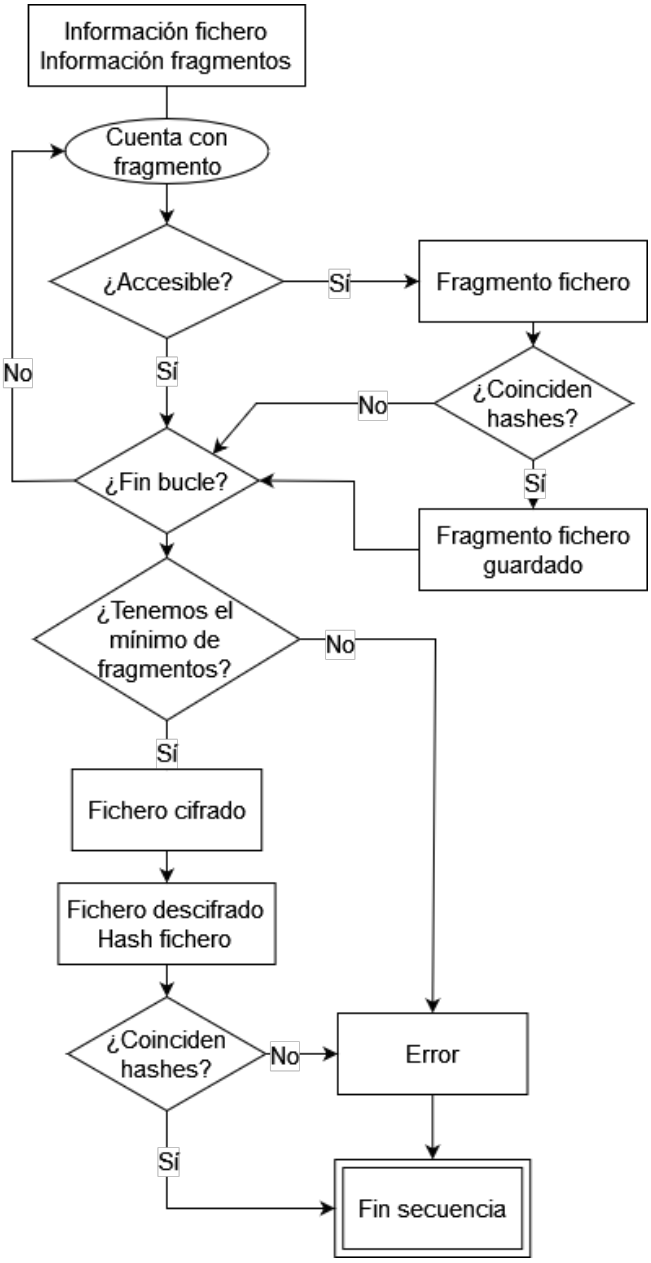

Figura 3.5: Diagrama lógico seguido para la descarga de un fichero

3.2.3. Borrar ficheros

La última operación que permite realizar la aplicación desarrollada consiste en eliminar un fichero de la nube y del agregador. Cuando el usuario solicita aplicar esta función a un fichero, el agregador comienza eliminándolo de su lista de ficheros, y por tanto de la base de datos. Al eliminar el fichero de la base de datos, se eliminan todos los datos de éste. Entre esto datos, se incluyen el hash del fichero, la clave con la que se cifró y el valor inicial empleado durante el cifrado. De esta forma, el fichero pasa a quedar irrecuperable por el agregador.

Además, eliminar el fichero de la base de datos, causa que todos los fragmentos generados a partir del fichero a borrar, tomen el valor nulo en el campo de fichero al que pertenecen. Es decir, estos fragmentos dejan de pertenecer a ningún fichero dentro de la base de datos. Esto permite obtener fácilmente de la base de datos una lista con todos los fragmentos pendientes de borrar.

Tras borrar el fichero, se genera una lista con todos los fragmentos que se tienen que eliminar del almacenamiento en la nube. La aplicación recorre esta lista tomando cada uno de los fragmentos e intentado eliminarlo de la nube. Si no se puede conectar con la cuenta en la que se almacena un fragmento, se omite el borrado de dicho fragmento. Éste permanecerá en la base de datos de la aplicación marcado para eliminarse. El borrado de este fragmento se volverá a intentar cuando se borre otro fichero, ya que se entiende que el problema de conectividad es temporal.

Si se consigue acceder al espacio en la nube donde se guardó el fragmento, se solicita su eliminación. En estos casos, tanto si se da un error en el borrado remoto como si no, se elimina el fragmento de entre los datos del agregador. La razón de esto, aún habiendo un error en la eliminación en la nube, es que se entiende el fallo de borrado como que el fichero ya no existe en la nube, es decir, ha sido descartado previamente.

En caso de que se intente eliminar un fichero sin conexión a internet, y por tanto sin poder acceder a las distintas cuentas, únicamente se eliminarán los datos del fichero en la base de datos. Gracias a esto, aún pudiéndose recuperar los distintos fragmentos, y con ello, el fichero cifrado, no se podría recuperar la clave y el valor inicial con el que descifrarlo. De esta forma, el fichero original debería permanecer inaccesible.

3.3. Base de Datos

La aplicación tiene que guardar de manera permanente información para su correcto funcionamiento. Estos datos consisten en las cuentas que se han vinculado, los ficheros que tiene que gestionar la aplicación y los fragmentos de estos ficheros.

La solución escogida para este desarrollo ha sido una base de datos local, propia del dispositivo en el que se ejecuta. La base de datos que emplea el agregador de nubes sigue el diagrama de entidad-relación [20] que se presenta en la Figura 3.6. A su vez, la información que representa cada uno de los atributos de una entidad, y por tanto los datos que guarda la base de datos, se encuentran detallados en la Tabla 3.10.

[20](2023) Wikipedia - Diagrama Entidad-Relación. [Online]. Available: `https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model`

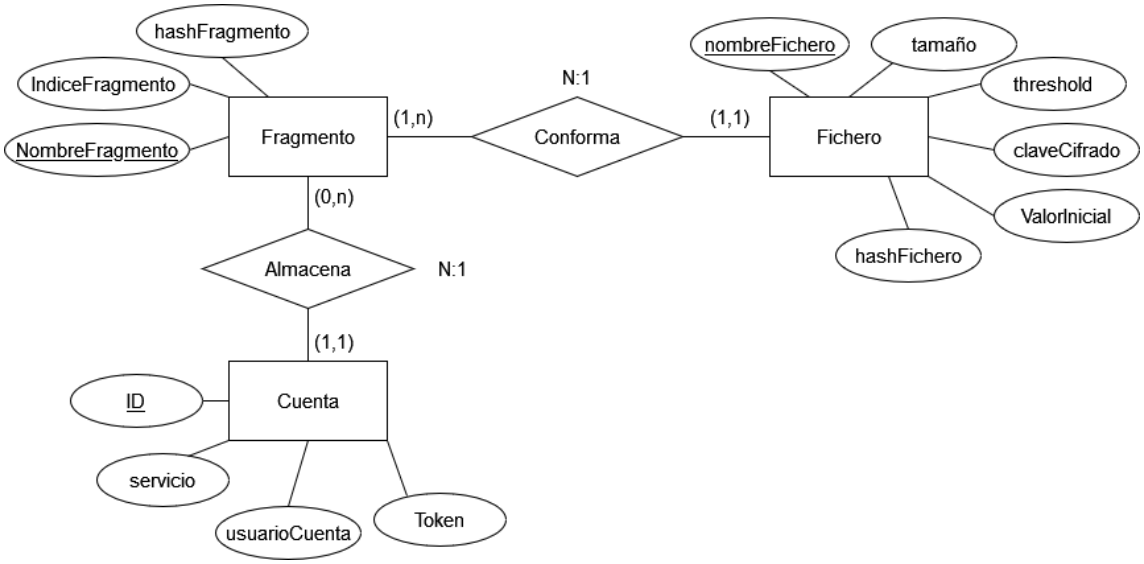

Figura 3.6: Diagrama E/R de la base de datos

Entidad	**Atributo**	**Tipo SQL**	**Descripción**
Cuenta	id	INT	Identificador de la nube dentro de la base de datos. Es un valor númerico único para cada entidad que se incrementa automáticamente, dentro de la base de datos, cada vez que se guarda una nueva cuenta.
	servicio	VARCHAR(32)	Servicio de almacenamiento en la nube al que pertenece la cuenta. Junto al atributo usuarioCuenta, se identifica la cuenta fuera de la base de datos.
	usuarioCuenta	VARCHAR(256)	Correo electrónico al que pertenece la cuenta. La tupla conformada por este correo electrónico y el servicio de almacenamiento en la nube, se emplean para identificar cada una de las cuentas.
	token	JSON	Token de acceso con el que establecer una conexión con la cuenta.
Fichero	nombreFichero	VARCHAR(256)	Nombre del fichero y su identificador dentro de la base de datos
	tamaño	INT	Tamaño del fichero antes de aplicar la división de secretos al fichero
	threshold	INT	Número de fragmentos necesarios para recuperar el fichero original
	claveCifrado	CHAR(128)	Clave empleada para cifrar el fichero
	valorInicial	CHAR(32)	Valor inicial empleado durante el cifrado del fichero
	hashFichero	CHAR(64)	Resultado hash del fichero antes de ser cifrado

Entidad	Atributo	Tipo SQL	Descripción
Fragmento	nombreFragmento	CHAR(64)	Nombre asignado a un fragmento e identificador de este dentro de la base de datos
	indiceFragmento	INT	Indice del fragmento en el esquema de Shamir
	hashFragmento	CHAR(64)	Resultado hash del fragmento
	id	INT	Clave externa de la entidad Cuenta. Identificador de la nube en la que se ha guardado el fragmento
	nombreFragmento	VARCHAR(256)	Clave externa de la entidad Fichero. Identificador del fichero que se puede recuperar con el fragmento

Cuadro 3.10: Atributos almacenados en la base de datos

De acuerdo a la tabla 3.10, cada uno de los fragmentos ocupa un máximo de trescientos noventa y dos (392) bytes. Por su parte, el espacio que se emplea para cada fichero en la base de datos es un máximo de cuatrocientos ochenta y ocho (488) bytes. Por último, el espacio necesario para guardar cada cuenta depende del tipo de ésta. Esto se debe a que cada proveedor de almacenamiento en la nube emplea tokens de acceso distintos, con longitudes diferentes.

Para la mayoría de servicios, el token de acceso suele ser un objeto JSON con el que no se superan los setecientos (700) bytes, por lo que el tamaño máximo de una cuenta en la base de datos, sería de aproximadamente mil (1000) bytes en estos casos. No obstante, el servicio OneDrive emplea tokens de acceso que serializados ocupan alrededor de los cinco mil seiscientos bytes (5.48 KB). Esto lleva a que el máximo tamaño que puede ocupar una cuenta sea de alrededor de los cinco mil novecientos bytes (5.78 KB).

3.3.1. Protección Base de Datos

La base de datos del agregador se almacena en la memoria del dispositivo como un fichero. Esto causa que la base de datos se encuentre disponible en el sistema de ficheros. Para evitar que los datos de la aplicación desarrollada puedan ser accedidos sin emplear el agregador de nubes, la base de datos se almacena cifrada.

Para poder trabajar con la base de datos, el agregador genera una copia descifrada en memoria RAM como una base de datos en memoria [21]. Estas bases de datos se montan mediante la función de sqlite `connect(nombre_BBDD)`, tomando el valor `':memory:'` como el nombre de la base de datos que se quiere abrir.

La llamada a la función `connect` de SQLite devuelve la conexión a la base de datos indicada. En el caso de las bases de datos en memoria, esta función retorna una conexión única a una base de datos con una caché privada. De esta

[21](2023) Bases de Datos SQLite en memoria. [Online]. Available: `https://www.sqlite.org/inmemorydb.html`

forma, si se llama a la función `connect(':memory:')` varias veces, cada una de las llamadas devolverá una conexión a una base de datos distinta del resto.

Cada vez que se establece conexión con una base de datos en memoria, se genera una nueva base de datos temporal. Cuando se cierra la conexión con una de estas bases de datos temporales, todos sus contenidos quedan inaccesibles. El espacio donde se guardan estos contenidos deja de estar reservado por SQLite, por lo que se pueden sobreescribir por cualquier proceso. Además, si se intenta conectar de nuevo a esta misma base de datos, la conexión genera una nueva base de datos vacía, en luagr de renovar dicha conexión.

El esquema que se sigue para proteger la base de datos toma características de la protección de volúmenes LUKS [22], de volúmenes VeraCrypt [23] y de la forma en la que la aplicación KeePass [24] almacena sus datos. Más concretamente, la base de datos se cifra a partir de una contraseña que introduce el usuario la primera vez que ejecuta el agregador, y por tanto cuando se crea la base de datos local. En el caso del agregador, esta base de datos es única al dispositivo en el que se ejecuta.

La contraseña del usuario no se guarda, sino que cada vez que se ejecuta se solicita su introducción para poder acceder a los contenidos de la base de datos. Cada vez que se introduce la contraseña, el agregador aplica una función de derivación de claves basada en el algoritmo PBKDF2 [25]. Esta función emplea como base el algoritmo *hash* SHA512 [26], y toma además de la contraseña un *salt* de treinta y dos bytes.

Este salt es generado de manera aleatoria, al crearse la base de datos. El *salt* consiste en una secuencia aleatoria que se guarda en claro, y con la que se modifica la salida de una función *hash*. Este salt se guarda junto a la base de datos como parte de una cabecera, para emplearse en la derivación de contraseñas en futuras ejecuciones del agregador.

Junto al salt, se genera también de manera aleatoria un valor inicial. Este valor inicial es el que se emplea para cifrar y descifrar la base de datos. Además, este valor inicial también se guarda en la cabecera la base de datos junto al salt.

El resultado de derivar la contraseña del usuario es una clave. Si esta clave se genera en la primera ejecución del agregador, es la clave de cifrado de la base de datos. Por el contrario, si se obtiene en otras ejecuciones de la aplicación, se tiene que comprobar si la clave es la de cifrado o no.

[22] C. Fruhwirth, "Luks1 on-disk format specification version 1.2.3," Tech. Rep.,2018.

[23] (2023) Cifrado de volúmenes veracrypt. [Online]. Available: https://veracrypt.eu/en/System%20Encryption.html

[24] (2023) Keepass security. [Online]. Available: https://keepass.info/help/base/security.html

[25] S. Josefsson, "Pkcs 5: Password-based key derivation function 2 (pbkdf2)test vectors," Internet Engineering Task Force (IETF), RFC 6070, jan 2011.[Online]. Available: https://datatracker.ietf.org/doc/html/rfc6070

[26] I. T. Laboratory, "Secure hash standard (shs)," FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, no. 180-4, aug 2015.

Para comprobar la clave de cifrado, en la primera ejecución del agregador se pasa la clave obtenida por la función hash SHA256. El resultado de esta función hash es un valor con el que comprobar si se ha obtenido la clave de cifrado en futuras ejecuciones del agregador. Este valor hash es el último componente de la cabecera de la base de datos cifrada (véase Figura 3.7).

Figura 3.7: Estructura fichero base de datos

Tras la primera ejecución del agregador, para comprobar si se ha obtenido la clave de cifrado, calcula el hash de la clave derivada. Si este valor hash coincide con el almacenado en la cabecera de la base de datos, se acepta que la clave derivada es la clave de cifrado. Por otro lado, si ambos hashes difieren, se entiende que la contraseña introducida no es correcta, por lo que se solicita al usuario que vuelva a introducir la contraseña.

Si se emplea la contraseña "`123`", por ejemplo, para proteger la base de datos, el agregador genera un salt aleatorio y deriva la contraseña. Para este ejemplo, el salt empleado es el presente en la tabla 3.11, indicado en rojo, y el resultado de la derivación es la secuencia de bytes marcada en verde en el cuadro 3.12. Además, el valor inicial generado, para este caso, es el indicado en la tabla 3.13 en azul.

ec	4c	c8	82	52	e6	05	4b	8d	2c	40	2a	26	79	4b	43
81	ed	59	e5	9f	7f	88	28	91	7c	14	84	d6	74	1f	c1

Cuadro 3.11: Salt derivación de claves.

a0	5d	5a	a4	df	07	82	6a	16	f6	12	88	e3	28	87	73
2a	89	9a	b7	49	8d	8a	ca	86	ae	1a	35	f4	ba	06	8a
70	fe	e5	12	57	a5	31	d5	ca	9d	66	ee	8b	ff	93	7e
37	4e	3e	5b	2d	14	1a	78	db	66	00	ed	8e	8b	59	e5

Cuadro 3.12: Resultado de la derivación de claves

67	3d	4e	dc	0b	df	9c	2d	ae	3c	22	80	1d	a3	0a	09

Cuadro 3.13: Ejemplo de valor inicial

En el cuadro 3.14 se indica el hash de la clave que se obtendría para la clave del ejemplo. La cabecera que se generaría en este caso, se encuentra definida en la tabla 3.15. Esta cabecera se compone por el salt de derivación, en rojo; el valor inicial, en azul, y el hash de la clave de cifrado, en naranja.

99	1b	d5	83	37	33	b8	7b	64	59	78	da	0f	e3	f1	75
8c	50	4e	d6	77	4a	07	7c	c7	cd	6c	61	b1	03	66	c3

Cuadro 3.14: Hash de la clave de cifrado

ec	4c	c8	82	52	e6	05	4b	8d	2c	40	2a	26	79	4b	43
81	ed	59	e5	9f	7f	88	28	91	7c	14	84	d6	74	1f	c1
67	3d	4e	dc	0b	df	9c	2d	ae	3c	22	80	1d	a3	0a	09
99	1b	d5	83	37	33	b8	7b	64	59	78	da	0f	e3	f1	75
8c	50	4e	d6	77	4a	07	7c	c7	cd	6c	61	b1	03	66	c3

Cuadro 3.15: Cabecera base de datos

Con la clave de cifrado, el agregador puede cifrar y descifrar la base de datos. Para ello, se emplea el algoritmo de cifrado AES256 [27], en modo CBC. Hay que resaltar que la clave de cifrado y por tanto la contraseña, no pueden ser cambiadas. No obstante, dicha funcionalidad puede ser incorporada en un futuro.

Por último, indicar que el descifrado se ejecuta solo al comienzo de la ejecución del agregador. Por el contrario, el cifrado se da tanto al finalizar la aplicación, como cada vez que se modifican los contenidos de la base de datos. Cada vez que se cifra la base de datos, se escribe en disco los contenidos cifrados. Esto se realiza para evitar que un cierre inesperado de la aplicación resulte en una pérdida de datos.

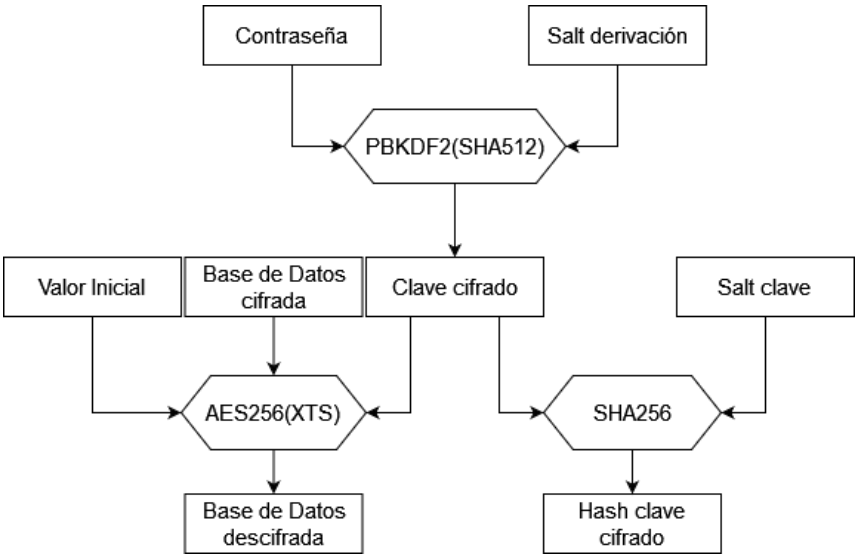

Figura 3.8: Esquema de claves para proteger la base de datos

[27] J. Daemen and V. Rijmen, The design of Rijndael: AES — the Advanced Encry- ption Standard. Springer-Verlag, 2002.

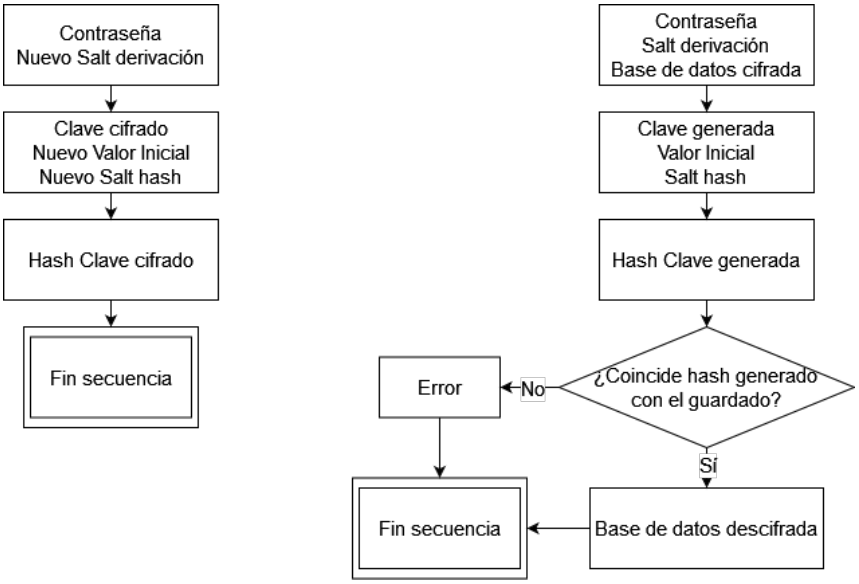

Figura 3.9: A la izquierda diagrama lógico seguido para la creación de las claves. A la derecha diagrama lógico seguido para el descifrado de la base de datos

Capítulo 4

Resultados y conclusiones

Si se analizan los resultados de este trabajo, se puede observar que se ha logrado cumplir con los objetivos marcados en su comienzo y descritos en el capítulo 1. En consecuencia, se ha creado una aplicación que guarda los datos de su usuario en la nube aumentando su disponibilidad. Además, la aplicación protege estos datos al almacenarlos en la nube, tal que el usuario es el único que puede acceder a sus datos.

La aplicación también comprueba la integridad de los datos del usuario, durante la recuperación de estos. De esta manera, si los datos han sido modificados, no permite su recuperación.

En primer lugar, se ha modificado el software *ssss* para que pudiese realizar correctamete la división de secretos a ficheros. Para ello, se ha desarrollado una librería C. Esta librería toma los contenidos del fichero, directamente desde disco. Los datos leidos del fichero se pasan al software modificado de *ssss* para fragmentarlos, mediante la división de secretos, o recuperar los datos originales. El resultado de operar los datos se devuelve a la librería, que los escribe directamente en disco. De este modo, la librería aplica la división de secretos sobre ficheros.

Por otro lado, se ha conseguido diseñar y programar el software encargado de gestionar varias cuentas en la nube, así como gestionar los ficheros de un usuario y almacenar datos en distintos espacios de almacenamiento en la nube. Este software puede ejecutarse como una aplicación a partir de una interfaz gráfica desarrollada en el marco Python multiplataforma Kivy [1].

La aplicación desarrollada permite al usuario vincular varias cuentas de almacenamiento en la nube, para tratarlas como un único servicio de almacenamiento. Además, permite al usuario subir a la nube ficheros, cifrándolos y aumentando su disponibilidad.

Para aumentar la disponibilidad de los ficheros del usuario, la aplicación aplica a los ficheros la división de secretos, haciendo uso de la librería ya comentada.

[1] (2023) Kivy. [Online]. Available: `https://kivy.org/`

Los fragmentos resultantes de la división de secretos, se distribuyen entre las distintas nubes vinculadas. Gracias a la división de secretos, para recuperar el fichero no es necesario descargar todos los fragmentos. Esto hace que se pueda recuperar el fichero, aún sin acceder a todas las cuentas.

Por otro lado, la aplicación también permite la descarga y borrado de ficheros. Estas operaciones solo pueden aplicarse a ficheros que han sido guardados en la nube haciendo uso del agregador. Durante la descarga, se comprueba la integridad del fichero descargado.

Por último, la aplicación hace uso de una base de datos, para poder guardar datos de manera persistente. Los datos a guardar en la base de datos son los que permiten conectarse con las cuentas vinculadas, así como los datos que permiten la descarga y recuperación de los ficheros.

La base de datos que emplea la aplicación, se almacena cifrada en el sistema de ficheros. La razón de esto es conseguir que solo el usuario, mediante la aplicación, pueda acceder a la información que se guarda de manera persistente.

Tras finalizar todo el desarrollo de este trabajo, se ha conseguido tener una versión completa y estable de la aplicación agregadora de nubes de almacenamiento.

4.1. Líneas futuras

Para futuras versiones de este proyecto, se plantean las siguientes mejoras.

El cambio principal que se propone es modificar la librería para asegurar su funcionamiento en sistemas operativos como puede ser Darwin, sistema operativo base de macOS e iOS. Esta mejora lleva implícita otra, que es la de hacer que el agregador de nubes no se trate solo de una aplicación de escritorio, sino que también sea una aplicación para dispositivos móviles. Este es el motivo por el cuál la aplicación se ha desarrollado en Python con una interfaz gráfica Kivy [2], ya que este tipo de interfaces gráficas pueden compilarse en aplicaciones Android e iOS.

Otra mejora, en este caso necesaria durante el tiempo de vida de la aplicación, es ir actualizando el uso de las APIs con las que conectar a los servicios en la nube, según se vayan actualizando. Todos estos cambios se tienen que realizar únicamente sobre los módulos con los que se establece conexión a las cuentas.

Por ejemplo, el servicio de almacenamiento en la nube Mega está desarrollando una SDK con la que conectarse a sus servicios. Esta SDK, a fecha de este proyecto, está diseñada para C++, aunque está planificado que también pueda emplearse con programas Python.

Debido a que la SDK para Python de Mega no está desarrollada, en este proyecto se ha empleado la anterior API de Python para Mega. Esta API pasó a estar

[2](2023) Kivy. [Online]. Available: `https://kivy.org/`

descontinuada el uno de marzo de 2023. En caso de problemas de compatibilidad, o de que MEGA complete la SDK para Python, se tendría que modificar exclusivamente el módulo de conexiones a las cuentas de MEGA.

En cuanto a funcionalidades a incluir, se proponen las siguientes:

1. Incluir un segundo modo de descarga, en el que el usuario indica el directorio donde guardar un fichero específico.
2. Realizar comprobaciones periódicas para el borrado de fragmentos pendientes de eliminar en remoto.
3. Permitir cambiar la contraseña de una base de datos.
4. Permitir el uso de varias bases de datos en un mismo equipo.
5. Añadir más servicios de almacenamiento en la nube que el agregador pueda emplear.
6. Incluir una configuración general para la división de los ficheros. El usuario tiene que poder modificar esta configuración.

Capítulo 5

Análisis de impacto

En el presente capítulo se presenta un análisis del posible impacto que puede tener el Trabajo Fin de Grado expuesto en este documento. Para ello, vamos a diferenciar y detallar los siguientes ámbitos de impacto.

A nivel personal, este trabajo me ha permitido aplicar varios de los conocimientos que he adquirido durante el grado. A su vez, la realización de este proyecto me ha dado la oportunidad de entrar en contacto con el mundo profesional de desarrollo software y de aprender acerca de métodos alternativos para la protección de información.

En lo social, se espera que esta aplicación permita guardar los datos protegidos a un mayor público, menos familiarizado con la criptografía. Confiamos en que el agregador de nubes desarrollado facilite a sus usuarios la gestión de sus ficheros en la nube. Este objetivo se espera que se consiga, gracias a que el agregador aumenta la disponibilidad de los datos. Para la recuperación de un fichero, no es necesario emplear todas las nubes usadas para su almacenamiento.

Por otro lado, también se espera que los datos de los usuarios se mantengan seguros, debido a que la aplicación los cifra antes de guardarlos en la nube. Este hecho hace que la información solo sea accesible por el usuario propietario de los datos, mediante la aplicación.

De este modo, esperamos que esta aplicación haga aumentar la cantidad de personas que protegen sus datos antes de guardarlos en remoto. Por ende también se espera que la cantidad de datos cuyo acceso está restringido a su propietario, se vea incrementada, a pesar del contenido que puedan conformar estos datos.

Por último, existen más ambitos los cuáles no consideramos que se vean afectados por este proyecto, por lo que no son mencionados.

No se ha hecho mención alguna con los objetivos de desarrollo sostenibles debido a que este trabajo no guarda relación directa alguna con ellos.

Bibliografía

[1] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[2] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on*. Los Alamitos, CA, USA: IEEE Computer Society, jun 1979, p. 313. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/AFIPS.1979.98

[3] G. Seroussi, "Table of low-weight binary irreducible polynomials," in *HP Labs Technical Reports*, HP, Ed., 1998.

[4] J. Daemen and V. Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[5] ——, *Mathematical preliminaries*. Springer-Verlag, 2002, ch. 2, pp. 4–8.

[6] D. Hardt, "The oauth 2.0 authorization framework," Internet Engineering Task Force (IETF), RFC 6749, oct 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

[7] T. Bray, "The javascript object notation (json) data interchange format," Internet Engineering Task Force (IETF), RFC 8259, dec 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8259

[8] (2023) Api python de mega. [Online]. Available: https://github.com/odwyersoftware/mega.py

[9] (2023) Api python de pcloud. [Online]. Available: https://pypi.org/project/pcloud/

[10] (2023) Api de google drive. [Online]. Available: https://developers.google.com/drive/api/reference/rest/v3?hl=es_419

[11] (2023) Api microsoft. [Online]. Available: https://learn.microsoft.com/en-us/graph/overview

[12] (2023) Api python de dropbox. [Online]. Available: https://dropbox-sdk-python.readthedocs.io/en/latest/

[13] I. T. Laboratory, "Secure hash standard (shs)," *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION*, no. 180-4, aug 2015.

[14] "Ieee standard for cryptographic protection of data on block-oriented storage devices," *IEEE Std 1619-2018 (Revision of IEEE Std 1619-2007)*, pp. 1–41, 2019.

[15] (2023) Cifrado de volúmenes veracrypt. [Online]. Available: https://veracrypt.eu/en/System%20Encryption.html

[16] C. Fruhwirth, "Luks1 on-disk format specification version 1.2.3," Tech. Rep., 2018.

[17] (2023) Keepass security. [Online]. Available: https://keepass.info/help/base/security.html

[18] S. Josefsson, "Pkcs 5: Password-based key derivation function 2 (pbkdf2) test vectors," Internet Engineering Task Force (IETF), RFC 6070, jan 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6070

[19] (2023) Kivy. [Online]. Available: https://kivy.org/

[20] (2023) Sqlite. [Online]. Available: https://www.sqlite.org/index.html

[21] (2023) In-memory. [Online]. Available: https://www.sqlite.org/inmemorydb.html

[22] (2023) Wikipedia - teorema de la interpolación. [Online]. Available: https://en.wikipedia.org/wiki/Polynomial_interpolation#Interpolation_theorem

[23] (2023) Wikipedia - interpolación polinómica de lagrange. [Online]. Available: https://en.wikipedia.org/wiki/Lagrange_polynomial

[24] (2023) Wikipedia - aritmética modular. [Online]. Available: https://en.wikipedia.org/wiki/Modular_arithmetic#Integers_modulo_n

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue May 30 18:53:58 CEST 2023
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)