Universidad de Almería

Escuela Politécnica Superior y Facultad de Ciencias Experimentales

Ingeniero en Informática

Un esquema de intercambio de claves y documentos cifrados para la plataforma Dropbox

Autor:
Óscar David Gómez López

Directores:
Juan Antonio López Ramos
José Antonio Álvarez Bermejo

8 de julio de 2014

A
mi familia

Agradecimientos

Quiero aprovechar estas primeras palabras para agradecer a todos los que me han apoyado y empujado a lo largo de este largo camino. Sin ellos seguramente no habría llegado hasta aquí.

Primero a mis padres. Que me han brindado comprensión tanto en los buenos y malos momentos. Que siempre me han apoyado incondicionalmente. Gracias por estar ahí.

A mis hermanos. Que me habéis ayudado de tantas maneras que me es imposible enumerarlas. Por cada una de ellas, muchas gracias. A mis sobrinos, por animarme una y otra vez con solo verlos sonreír. También a mis cuñadas, por enseñarme que hay lazos tan fuertes como la sangre.

A mis compañeros de clase. Los que me han aguantado durante estos cinco años. En cada cuatrimestre. En cada práctica. En cada examen. Y en cada café.

A mis amigos de toda la vida de Vélez blanco. A los que he hecho durante estos años en Almería. A mis compañeros de kendo. A mis compañeros de japonés. Gracias por ayudarme a desconectar y enseñarme que siempre se puede ver la vida de otra forma.

Y por último a mis directores. Por todo su esfuerzo y dedicación durante las incontables horas necesarias para realizar este proyecto.

Gracias.

Índice general

Índice de figuras

•

Índice de tablas

Códigos

Capítulo 1

Introducción

1.1. ¿Qué es la Criptología?

La criptología es la disciplina científica que se dedica al estudio de códigos secretos con el fin de que un mensaje sea imposible de leer por terceros no autorizados.

El estudio de la criptología se puede dividir en diversos campos:

- **Criptografía**: su propósito es proporcionar seguridad a las comunicaciones. En un principio se aplicaba principalmente a los mensajes escritos, mas sus leyes y métodos se aplican igualmente al flujo de datos entre ordenadores.
- **Criptoanálisis**: su propósito es buscar las vulnerabilidades de la criptografía,es decir, conseguir obtener el significado de los mensajes construidos mediante la Criptografía sin tener autorización para ello.

Ambos campos, criptografía y criptoanálisis, podrían ser considerados opuestos, pero están profundamente relacionados, pues la existencia del criptoanálisis lleva a que cada vez se desarrollen criptosistemas mas complejos y seguros.

1.2. Objetivos de la criptografía

La criptografía se propone cumplir los siguientes principios:

- **Confidencialidad**: La información es solo accesible o revelada a individuos o entidades autorizados.
- **Autentificación**: Asegura la identidad de un individuo, es decir, podemos garantizar que un individuo es quien dice ser.
- **Integridad**: Asegura que los datos no se han modificado o destruido de forma no autorizada.
- **No Repudio**: Se puede probar que los participantes de una transacción realmente la autorizan y que no pueden negar de ninguna forma su participación.

Figura 1.1: Criptosistema simétrico

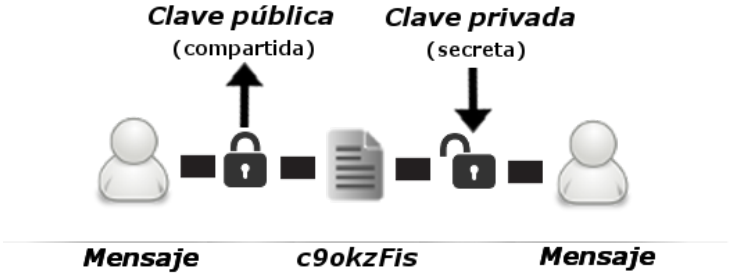

Figura 1.2: Criptosistema asimétrico

1.3. Tipos de criptosistemas

En la criptografía existen principalmente dos tipos de criptosistemas:

- **Simétricos**: utilizan la misma clave para cifrar y descifrar.
- **Asimétricos**: utilizan una clave para cifrar(la clave publica) y otra para descifrar(la clave privada).

Los criptosistemas simétricos suelen ser mas eficientes debido a la mayor simplicidad de sus algoritmos de cifrado y descifrado. En cambio, los algoritmos de clave asimétrica suelen requerir mayor trabajo, son mas lentos, debido a la complejidad de los problemas matemáticos en los que se basan. Se suele recurrir a los simétricos para encriptar mensajes o archivos grandes.

El riesgo de los criptosistemas simétricos es la revelación, o filtrado, de la clave, pues con esta podrá descifrar todas las comunicaciones. Esto no sucede con los asimétricos, pues con la clave publica solo podrá cifrar información y nunca descifrarla. La clave privada nunca se comparte con nadie.

Otro uso de los criptosistemas asimétricos es la firma digital. Ésta permite al receptor confirmar la identidad del emisor de forma segura y asegurarse que no ha sido modificado por terceros. El emisor tampoco podrá negar ser el autor de la información.

1.4. Problema del intercambio de claves

El problema consiste en que dos usuarios, que no han tenido un contacto previo quieren decidir una clave secreta sobre un medio inseguro para mantener una comunicación segura entre ellos. Esta clave secreta suele ser simétrica por lo que hemos comentado anteriormente.

Este problema se extiende a comunicaciones entre más de dos usuarios. Este tipo de esquemas se conocen habitualmente como esquemas multidifusión de claves. Estos buscan la eficiencia desde el punto de vista computacional y la seguridad de los mismos y se pueden encontrar en cualquier comunicación que involucre a un grupo de personas, ya sea una comunicación de todos hacia todos, como podría tratarse del caso de una multiconferencia, o el caso de una comunicación de uno a grupo, como podría ser el caso, por ejemplo, de una emisión de cualquier información de tipo multimedia bajo demanda.

Los modelos de intercambio multidifusión[9] se dividen en tres categorías, dependiendo de las autoridades encargadas de la generación, regeneración, distribución y redistribución de claves en estos modelos:

- **Centralizados**, son los modelos que tienen una autoridad central que se encarga de los cambios de la clave de sesión del esquema.
- **Descentralizados**, aquellos modelos donde existe un grupo de usuarios independientes que actúan como autoridades locales para la distribución de claves.
- **Distribuidos**, donde los propios miembros llevan a cabo la distribución y generación de claves de forma coordinada.

Desde que Diffie y Hellmann [2] propusieran el primer algoritmo de intercambio de claves, se puede encontrar una bibliografía muy extensa y exhaustiva relacionada con este problema. La mayoría de protocolos propuestos basan su aritmética sobre estructuras algebraicas conmutativas. Entre estos destacan principalmente dos problemas:

- El problema de la factorización entera: que consiste en la dificultad de factorizar un número grande sobre el anillo de los enteros. El criptosistema RSA[10] basa su fortaleza en la dificultad de resolver este problema.
- El problema del Logaritmo Discreto: sobre el cuerpo finito Zp con p un primo grande. El conocido protocolo ElGamal[4] y todas sus numerosas variantes basan su fortaleza en este problema.

Estos problemas actualmente resultan inabordables computacionalmente, pero parece una idea aceptada que pueden resultar inseguros a corto o medio plazo, debido al constante e imparable aumento de la potencia de computación de los ordenadores y estaciones de trabajo actuales. Como demuestra teóricamente Shor en su trabajo[11] el protocolo RSA, ElGamal y Diffie-Hellman resultaran inseguros, por tanto la tendencia actual se encamina hacia algoritmos basados en otras estructuras como las curvas elípticas[5][6] o sobre problemas algebraicos más complicados como el que se trata en [1].

1.5. Propósito del proyecto

Este proyecto se basa en los resultados de [1] sobre la multidifusión de claves usando estructuras algebraicas no conmutativas: las matrices. Siendo los protocolos que propone(multidifusión secuencia y multidifusión en bloque) del tipo distribuido.

Haciendo uso de esto se pretende desarrollar una aplicación de intercambio de archivos cifrados donde la clave sea decidida por los usuarios en el momento. El intercambio sera efímero, solo manteniéndose mientras haya usuarios en la conferencia.

Para realizar esto existen varias alternativas. La primera de ellas consiste en la existencia de un servidor central sobre el cual se realiza el intercambio de claves y donde se almacenan los archivos cifrados. La segunda alternativa consiste en reducir el papel del servidor central únicamente a poner en contacto a los conferenciantes y estos ya realizarían entre ellos el intercambio de claves y el compartimento de los archivos.

Este proyecto implementara la primera alternativa, usando Dropbox como servidor central. Esto supondrá una abstracción que facilitara el proceso, así como proporcionar el medio perfecto para el intercambio de archivos.

1.6. Estructura del proyecto

La estructura de los capítulos que componen este proyecto son:

- **Capitulo 1. Introducción:** En el primer capítulo se va a presentar el proyecto. Se introducirán diversos conceptos criptográficos, así como la motivación y propósito del proyecto.
- **Capitulo 2. Protocolos de multidifusión de claves:** En este capitulo se introducirá la base matemática necesaria junto con dos protocolos de multidifusión de claves.
- **Capitulo 3. Implementación técnica de la solución:** En el tercer capitulo se hablará de todo lo relacionado con la implementación de los protocolos. La tecnología usada, el diseño de la aplicación y los aspectos más relevantes de la implementación.
- **Capitulo 4. Casos de uso:** En el cuarto capítulo se realizan pruebas practicas de las diferentes funcionalidades de la aplicación. Con el objetivo de comprobar el correcto funcionamiento de la aplicación, así como hacer de guía de la misma.

Capítulo 2

Protocolos de multidifusión de claves

2.1. Preliminares

En este capítulo se van a describir ambos protocolos de multidifusión de claves, pero antes resulta necesario introducir una serie de elementos. Todos estos elementos, así como los protocolos son extraídos de [1].

2.1.1. El anillo $E_p^{(m)}$

Primero se van a introducir cómo son los elementos y operaciones, suma y producto, mediante los cuales vamos a hacer los cálculos, el anillo $E_p^{(m)}$. Se denota por $Mat_m(\mathbb{Z})$ el conjunto de las matrices de tamaño m x m con elementos en $\mathbb{Z}$.

Teorema: Supongamos que p es un número primo y m ≥ 2 es un entero.

El conjunto

$$E_p^{(m)} = \{[a_{ij}] \in Mat_m(\mathbb{Z}) \ \mid a_{ij} \in \mathbb{Z}_{p^i} \ si \ i \leq j, \ a_{ij} \in p^{i-j}\mathbb{Z}_{p^j} \ si \ i > j\}$$

es un anillo unitario y no conmutativo con la adición y la multiplicación definidas, respectivamente, como

$$[a_{ij}] + [b_{ij}] = [(a_{ij} + b_{ij}) mod p^i]$$

$$[a_{ij}] \cdot [b_{ij}] = \left[\left(\sum_{k=1}^{m} a_{ik} b_{kj} \right) mod p^i \right]$$

Como se puede ver en el teorema los elementos del anillo, los cuales son los elementos posteriormente usados en el protocolo, son matrices cuadradas cuyos elementos siguen la estructura

vista en el teorema, donde los elemento de la primera fila estarán en $\mathbb{Z}_p$, los de la segunda en $\mathbb{Z}_{p^2}$, los de la tercera en $\mathbb{Z}_{p^3}$ y así sucesivamente.

Elementos que utilizaremos en los protocolos de una forma básica para su correcto funcionamiento son los elementos del centro del anillo $E_p^{(m)}$, es decir, aquellos elementos que conmutan con cualquier otro al hacer el producto.

Lema: El centro de $E_p^{(m)}$ es el conjunto

$$Z\left(E_p^{(m)}\right) = \{[a_{ij}] \in Mat_m(\mathbb{Z}) \mid a_{ii} = \sum_{j=0}^{i-1} p^j x_j,\ con\ x_j\ \in \mathbb{Z}_p\ y\ a_{ij} = 0\ si\ i \neq j\}$$

y, en consecuencia, el número de elementos del mismo es Card $\left(Z\left(E_p^{(m)}\right)\right) = p^m$

2.1.2. Anillo de polinomios $Z\left(E_p^{(m)}\right)[X]$

El anillo de polinomios se define como el conjunto de polinomios con coeficientes en el anillo base. De este modo se tiene que

$$Z\left(G_p^{(m)}\right)[X] = \{\sum_{i=0}^{n} g_i X^i :\ n \in I\!N \text{ y } g_i \in G_p^{(m)} \forall i = 1, \ldots, n\}$$

Si k y l ∈ N, aunque este sea un anillo no conmutativo, tal y como se prueba en [1] se cumple que

$$f(M)^k g(M)^l = g(M)^l f(M)^k,\ para\ todo\ M \in R$$

Esta propiedad es la idea central que permite definir los siguientes protocolos de intercambio de claves.

2.2. Elementos del protocolo

En ambos protocolos existen necesariamente dos partes para poder decidir la clave secreta compartida por todos los usuarios: la clave privada de cada usuario y la clave publica de la conferencia.

La parte publica de la conferencia está compuesta por dos elementos: M y N ∈ $E_p^{(m)}$. Por su parte la privada de cada usuario esta compuesta por:

- **f(x):** un polinomio del anillo de polinomios Z(R)[X], que sera el elemento principal de la parte privada.

- **r y s:** dos números enteros.

Por último, los elementos que el protocolo genera durante el intercambio:

- $f(M)$**:** resultado de aplicar la función privada de un usuario i a la matriz publica M.
- K_i**:** un elemento de $E_p^{(m)}$ usado durante el intercambio.
- L_i**:** un elemento de $E_p^{(m)}$ usado durante el intercambio.
- S_i**:** un elemento de $E_p^{(m)}$. Es el secreto compartido por todos los usuarios, es decir, todos los S_i deben tener el mismo valor.

Después de realizar el intercambio y obtener S_i se utilizara un criptosistema simétrico para realizar la comunicación, usando para esto algoritmos tales como AES o DES.

2.3. Protocolo de multidifusión sencuencial

Protocolo de multidifusión secuencial: Cada uno de los usuarios U_i para i=1,2,...h elige un polinomio f_i(X) ∈ Z(R)[X] y un par de enteros positivos r_i y s_i. De esta forma la clave privada de cada usuario U_i está formada por la terna (f_i(X), r_i, s_i).

Consideramos el elemento público K_0 = N ∈ R Z(R).

El esquema sigue los siguientes pasos:

(a) El usuario U_1 calcula el elemento K_1 de R como

$$K_1 = f_1(M)^{r_1} K_0 f_1(M)^{s_1}$$

A continuación el usuario U_1 envía el elemento K_1 al usuario U_2.

(b) El usuario U_2 calcula el elemento K_2 ∈ R como

$$K_2 = f_2(M)^{r_2} K_1 f_2(M)^{s_2}$$

de forma que el usuario U_2 envía al usuario U_3 el vector (K_1,K_2) de elementos de R.

(c) Generalizando el proceso, para i=3,4,...,h-2,h-1 el usuario U_i calcula el elemento

$$K_i = f_i(M)^{r_i} K_{i-1} f_i(M)^{s_i} \in R$$

Entonces, el usuario U_i envía al usuario U_{i+1} el vector (K_1,K_2,...,K_{i-1},,K_i,) de elementos de R.

(d) Cuando el usuario U_h recibe el vector (K_1,K_2,...,K_{h-2},,K_{h-1},) calcula el elemento

$$L_l^{(h)} = f_h(M)^{r_h} K_l f_h(M)^{s_h} \in R$$

con l = 0,1,2,...,h-2,h-1

Entonces el usuario U_h envía al usuario U_{h-1} el vector ($L_0^{(h)}$, $L_1^{(h)}$,..., $L_{h-3}^{(h)}$, $L_{h-2}^{(h)}$) de elementos de R.

(e) Cuando el usuario U_{h-1} recibe el vector ($L_0^{(h)}$, $L_1^{(h)}$,..., $L_{h-3}^{(h)}$, $L_{h-2}^{(h)}$) del usuario U_h, entonces calcula el elemento

$$L_l^{(h-1)} = f_{h-1}(M)^{r_{h-1}} L_l^{(h)} f_{h-1}(M)^{s_{h-1}} \in R$$

con l = 0,1,2,...,h-3,h-2

El usuario U_{h-1} envía al usuario U_{h-2} el vector ($L_0^{(h-1)}$, $L_1^{(h-1)}$,..., $L_{h-4}^{(h-1)}$, $L_{h-3}^{(h-1)}$) de elementos de R.

(f) Este proceso se generaliza para i=2,3,...,h-2,h-1, de forma que cuando el usuario U_{h-i} recibe del usuario U_{h-i+1} el vector de elementos R dado por

$$(L_0^{(h-i+1)}, L_1^{(h-i+1)}, ..., L_{h-2}^{(h-i+1)}, L_{h-1}^{(h-i+1)})$$

calcula el elemento

$$L_l^{(h-i)} = f_{h-i}(M)^{r_{h-i}} L_l^{(h-i+1)} f_{h-i}(M)^{s_{h-i}}$$

con l=0,1,2,...,h-i-2,h-i-1.

Seguidamente el usuario U_{h-1} envía el vector ($L_0^{(h-i)}$, $L_1^{(h-i)}$,..., $L_{h-i-3}^{(h-i)}$, $L_{h-i-2}^{(h-i)}$) de elementos de R al usuario U_{h-i-1}.

(g) Por último cada uno de los usuarios U_{h-i} para i=0,1,2,...,h-2,h-1, ha calculado el elemento S_{h-1} = $L_{h-i-1}^{(h-i)} \in$ R. Notemos que éste es el único elemento que no ha sido envidado al usuario U_{h-i-1}, siendo este el secreto compartido por todos los usuarios.

2.4. Protocolo de multidifusión en bloque

Antes de mostrar el funcionamiento del siguiente esquema de multidifusión, necesitamos introducir la siguiente notación para describir y simplificar ciertos sumatorios.

$$\sigma(j,1,i) = \sum_{j=1}^{i-1} j \quad \text{y} \quad \sigma(j,1,i,l) = \sum_{\substack{j=1 \\ j \neq i-l}}^{i-1} j$$

Protocolo de multidifusión en bloque: Consideramos que se comparten los elementos públicos, M $\in$ R y K_0 = N $\in$ R Z(R). Cada usuario U_i, para i = 1,2,...,h elige un polinomio f_i (X) $\in$ Z(R)[X] y un par de enteros positivos r_i y s_i. De esta forma la clave privada de cada usuario U_i está formada por la terna (f_i(X), r_i, s_i).

(a) El usuario U_1 calcula el elemento $K_1 \in$ R como

$$K_1 = f_1(M)^{r_1} K_0 f_1(M)^{s_1}$$

El usuario U_1 envía el elemento K_1 al usuario U_2.

(b) El usuario U_2 calcula el elemento K_2 y K_3 de R

$$K_2 = f_2(M)^{r_2} K_0 f_2(M)^{s_2}$$

$$K_3 = f_2(M)^{r_2} K_1 f_2(M)^{s_2}$$

El usuario U_2 envía al usuario U_3 el vector (K_1,K_2,K_3) de elementos de R.

(c) El usuario U_3 calcula los elementos K_4 , K_5 y K_6 como

$$K_4 = f_3(M)^{r_3} K_1 f_3(M)^{s_3}$$

$$K_5 = f_3(M)^{r_3} K_2 f_3(M)^{s_3}$$

$$K_6 = f_3(M)^{r_3} K_3 f_3(M)^{s_3}$$

El usuario U_3 envía al usuario U_4 el vector (K_3,K_4,K_5,K_6) de elementos de R.

(d) En general, con la notación vista en la expresión de arriba, para i = 4,5,...,h-1, el usuario U_i calcula los elementos de R como

$$K_{i+\sigma(j,1,i,l)} = f_i(M)^{r_i} K_{\sigma(j,1,i,l)} f_i(M)^{s_i}, \quad para \ = 1, 2, 3, ..., i-1$$

$$K_{i+\sigma(j,1,i)} = f_i(M)^{r_i} K_{\sigma(j,1,i)} f_i(M)^{s_i}$$

El usuario U_i envía al usuario U_{i+1} el (i+1)-vector de elementos de R

$$(K_{i-1+\sigma(j,1,i,1)},\ K_{i+\sigma(j,1,i,1)},\ K_{i+\sigma(j,1,i,2)}, ...,\ K_{i+\sigma(j,1,i,i-1)},\ K_{\sigma(j,1,i+1)})$$

(e) Cuando el usuario U_h recibe el h-vector

$$(K_{h-2+\sigma(j,1,h-1,1)},\ K_{h-1+\sigma(j,1,h-1,1)},\ K_{h-1+\sigma(j,1,h-1,2)}, ...,\ K_{h-1+\sigma(j,1,h-1,h-2)},\ K_{\sigma(j,1,h)})$$

calcula los siguientes elementos del anillo R

$$L_1^{(h)} = f_h(M)^{r_h} K_{h-2+\sigma(j,1,h-1,1)} f_h(M)^{s_h}$$

$$L_l^{(h)} = f_h(M)^{r_h} K_{h-1+\sigma(j,1,h-1,l)} f_h(M)^{s_h}$$
$$= f_h(M)^{r_h} K_{\sigma(j,1,h,l)} f_h(M)^{s_h} \quad para \ = 2, 3, ..., h-1$$

$$L_h^{(h)} = f_h(M)^{r_h} K_{h-2+\sigma(j,1,h-1,1)} f_h(M)^{s_h}$$

El último usuario U_h envía a cada uno de los anteriores usuarios de forma simultánea el (h-1)-vector $(L_1^{(h)},L_2^{(h)},...L_{h-1}^{(h)})$ de elementos de R.

(f) Finalmente, cuando el usuario U_i, para i=1,2,...,h-1, recibe el (h-1)-vector $(L_1^{(h)},L_2^{(h)},...L_{h-1}^{(h)})$, este toma la (h-i)-componente del vector $L_{h-i}^{(h)}$, y calcula el elemento

$$S_i = f_i(M)^{r_i} L_{h-i}^{(h)} f_i(M)^{s_i}$$

Por último y por cuestiones de notación, el usuario U_h denota por S_h el elemento $L_h^{(h)}$. Siendo S el secreto compartido por todos los usuarios.

2.5. Comparativa de los protocolos

En este punto se compararan las principales diferencias de los dos protocolos vistos: multidifusión secuencia (figura 2.1) y multidifusión en bloque (figura 2.2).

El protocolo de difusión en bloque resulta mas eficiente que el secuencial en casi todos los aspectos.

Primero requiere una menor cantidad de mensajes para decidir la clave. El protocolo secuencia requiere $2(h-1) = 2h-2$ mensajes, (h-1) para mandar los vectores K y, otros, (h-1) para mandar los vectores L. Mientras que el protocolo en bloque requiere de $(h-1)+1 = h$, (h-1) para mandar los vectores K y un mensaje ,de broadcast, para el vector L.

También tiene una menor carga computacional y por tanto de un menor tiempo para decidir la clave secreta compartida. El protocolo secuencial requiere de calcular el vector L tantas veces como usuarios, mientras que el protocolo en bloque una única vez- al solo tener que calcularlo el último usuario -.

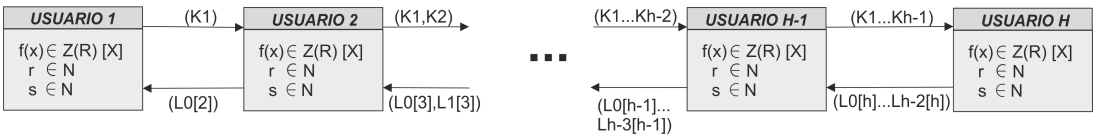

Figura 2.1: Protocolo de multidifusión de claves secuencial

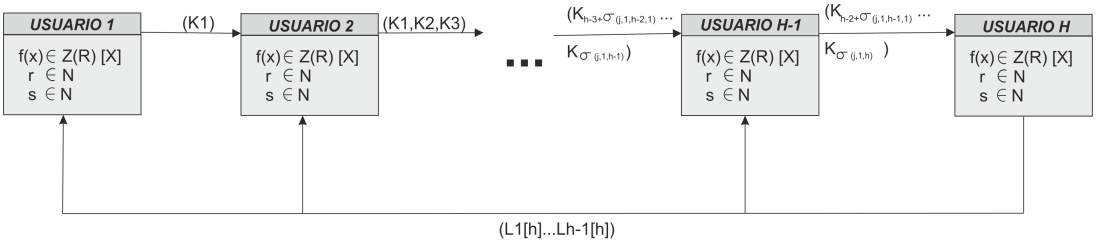

Figura 2.2: Protocolo de multidifusión de claves bloque

2.6. Unir y expulsar conferenciantes

Añadir o expulsar a un usuario en una conferencia existente es un elemento importante del protocolo. En una conferencia con muchos usuarios, estos estarán en movimiento pudiendo en cualquier momento existir la necesidad de unir un nuevo usuario a la conferencia, expulsar a algún usuario que deje de ser de confianza o,simplemente, abandonar la conferencia. En todos estos escenarios es necesario actualizar la clave secreta compartida. Una posibilidad sería regenerar desde cero la conferencia con los nuevos usuarios, pero seria lento y se repetirían cálculos previamente realizados. Para evitar esto, se introducen las siguientes metodologías donde solo se calcularan los datos indispensables.

2.6.1. Unir conferenciante

Para unir, o añadir, un nuevo usuario a la conferencia esté se posicionara al final de la misma. El usuario h, el que anteriormente era el último, cambiara su clave privada y mandara el vector de elementos K_i al nuevo usuario, esté continuara de una forma u otra dependiendo del protocolo: mandando el vector de elementos L_i al anterior último (figura 3.9) y así secuencialmente hasta el primero; o mandándolo a todos los usuarios a la vez (figura 2.4).

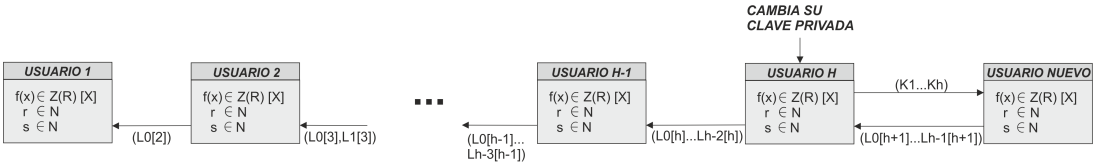

Figura 2.3: Unir usuario - Protocolo de multidifusión de claves secuencial

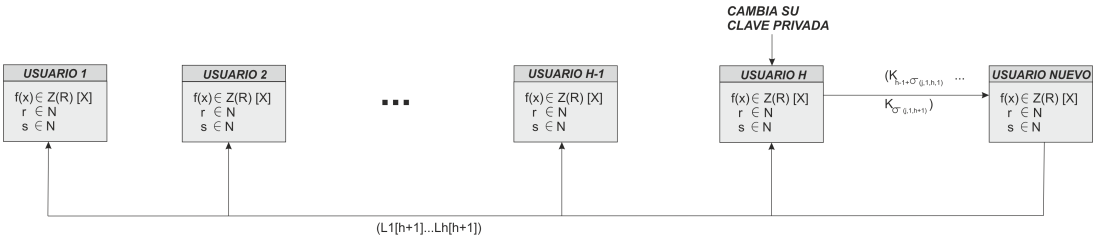

Figura 2.4: Unir usuario - Protocolo de multidifusión de claves bloque

2.6.2. Expulsar conferenciante

Para expulsar a un usuario de una conferencia, o cuando esté la deja voluntariamente, sera necesario que el resto de usuarios actualicen su clave compartida secreta. El usuario anterior al que abandona la conferencia cambiara su clave privada y mandara el vector de elementos K_i al usuario siguiente del que abandono la conferencia- si no existiese un usuario siguiente ,porque el usuario que abandono fuera el último, se ignoraría esté paso y se pasara a mandar el vector de elementos L_i -. Los usuarios seguirán mandando su vector de elementos K_i hasta llegar al último usuario, esté continuara de una forma u otra dependiendo del protocolo: mandando el vector de elementos L_i al anterior último (figura 2.5) y así secuencialmente hasta el primero; o mandándolo a todos los usuarios a la vez (figura 2.6).

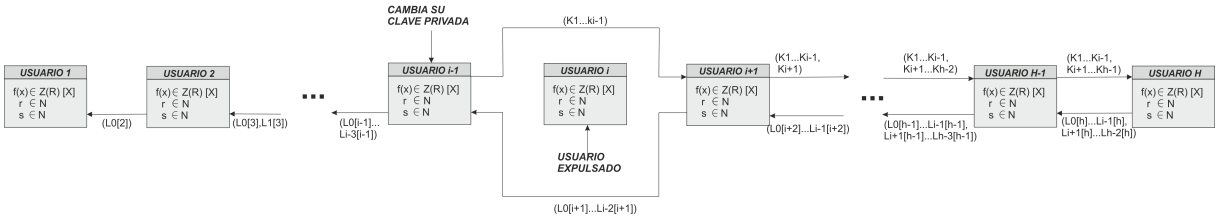

Figura 2.5: Expulsar usuario - Protocolo de multidifusión de claves secuencial

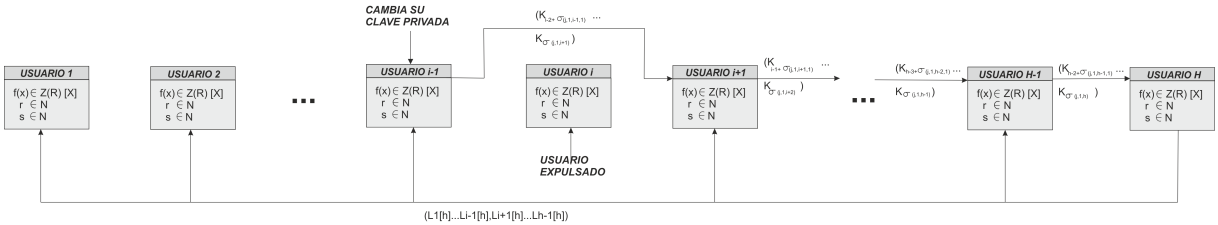

Figura 2.6: Expulsar usuario - Protocolo de multidifusión de claves bloque

2.7. Seguridad

La seguridad de los criptosistemas basados en plataformas criptográficas no conmutativas se basan en diferentes problemas. Los protocolos vistos en este capitulo se basan en **el problema de la descomposición o DP (Descomposition Problem)**. Esté consiste en dados (x,y) ∈ G x G y $m, n \in \mathbb{Z}$ y S ⊆ G, el problema consiste en encontrar z_1, z_2 ∈ S tales que y=z_1xz_2. Para él que no se conoce ningún algoritmo probabilístico que sea capaz de resolver el problema en tiempo polinómico.

La seguridad de ambos protocolos es equivalente a resolver un problema de este tipo ya que dados M y N públicos, a través del canal se envían elementos de la forma K=$f(M)^r$N$g(M)^s$ y entonces, un atacante debería determinar dos matrices A y B tales que K=ANB.

Un atacante que intentase descubrir el secreto compartido por fuerza bruta debería de obtener la clave privada de algún usuario, es decir, encontrar los polinomios f(X) y g(X), así como los enteros r y s. De este modo y para hacernos una idea del número de posibilidades, el número de posibles polinomios de un cierto grado n con coeficientes en el centro del anillo $G_p^{(m)}$ es de (n+1)p^m, lo que puede dar lugar a números realmente grandes para una adecuada elección del primo p y del tamaño de las matrices m.

A modo de ejemplo, notemos que es suficiente tomar un primo p de 7 cifras como es p = 1150249, m = 20 y polinomios de grado n = 5 para alcanzar del orden de 10121 posibles polinomios y mantener, por tanto, el nivel de seguridad.

Capítulo 3

Implementación técnica de la solución

3.1. Selección de tecnología

El primer paso para realizar la implementación será definir el cómo de la misma, decidir qué tecnología y técnicas se van a utilizar.

En este proyecto se va a construir una plataforma de intercambio seguro de ficheros mediante a la utilización de los protocolos vistos en el capitulo anterior. En la que se puedan garantizar:

- Intercambio seguro de ficheros en grupos de usuarios.
- La clave secreta con la que se cifrarán y descifrará será decidida por los participantes en el momento de iniciar una conferencia y solo será conocida por ellos.
- Se podrán unir y expulsar participantes de la conferencia sin tener que volver a generar la clave desde cero.
- Las conferencias serán efímeras y una vez terminen no quedará ningún rastro de que existieron.

Para esto se utilizará el lenguaje de programación Java. Es un lenguaje orientado a objetos y multiplataforma por lo que la aplicación podrá ser ejecutada tanto en sistemas Windows, Linux o Mac. Ademas Java también proporciona una relativamente fácil migración a sistemas móviles basados en Android. También posee el framework: *Java Cryptography Architecture (JCA)* que será una parte fundamental en la labor criptográfica.

Por otro lado esta la comunicación se implementará haciendo uso de la API de DropBox. Esta proporcionará un punto de encuentro donde todos los usuarios que utilicen la aplicación podrán intercambiar información. Tanto aquellos mensajes necesarios para decidir la clave secreta, como los ficheros que sean intercambiados. Los ficheros intercambiados estarán cifrados en DropBox con la clave secreta, decidida entre los participantes, y no podrá ser deducida a partir de la información intercambiada sobre DropBox.

Por último se utilizarán una serie de herramientas enfocadas a agilizar las tareas de desarrollo, gestión y pruebas. Eclipse, un IDE. Subversion, una herramienta de control de versiones. JUnit un framework de java para realizar pruebas unitarias.

3.1.1. Java

Se trata de un lenguaje de programación de propósito general orientado a objetos que pueden ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Entre sus ventajas se encuentra la similitud de la maquina virtual de Java y Android(Dalvik). La DVM no afirma ser una máquina virtual de Java debido a que le ocasionaría problemas de licenciamiento, aunque sin embargo cumple ese propósito. Ésto permite que sea posible reutilizar casi siempre — salvo algunas excepciones seria en el trabajo con elementos gráficos y de red— código desarrollado para Java en Android, especialmente aquel encargado de la gestión de clases y cálculos.

La reutilización de código entre Java y Android resulta muy conveniente pues la filosofía de los protocolos se adapta fácilmente a entornos móviles, tal como podría ser un sistema de mensajería seguro y, al hacer el desarrollo en Java, se dejaría la puerta abierta a desarrollos futuros en Java o sobre dispositivos Android. Es interesante que se pueda aprovechar desde el diseño a la implementación, sin más que usar compiladores cruzado a nivel de bytecode.

Otra ventaja es la existencia de un gran número de librerías que ofrecen una amplia diversidad de funcionalidades. Entre la que podemos encontrar la librería : Java Cryptography Architecture (JCA)[8]. Esta librería es un framework para trabajar con criptografía usando el lenguaje de programación Java.

Pero no todo son ventajas, una máquina virtual es un intermediario entre las aplicaciones y el sistema. Añadir un intermediario hace que las tareas requieran una mayor cantidad de recurso, pues, la máquina virtual, requiere recursos hardware del sistema para funcionar e, igualmente, las aplicaciones requerirán de mas recursos hardware. Por otro lado, las soluciones nativas requieren únicamente los recursos del sistema pertinentes a la aplicación.

3.1.2. JUnit

Es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck destinadas a automatizar las pruebas unitarias de aplicaciones Java.Esto permite asegurar que cada uno de los módulos funcione correctamente por separado.

En este proyecto su uso resulta realmente conveniente para asegurar el correcto funcionamiento de toda la base matemática necesaria para la implementación de los protocolos, así como para comprobar el funcionamiento de estos.

3.1.3. Subversion

Se trata de un sistema de control de versiones que permite gestionar los diversos cambios que se realizan en un elemento. Ésta lleva un histórico de todos los cambios realizados facilitando así la administración de todas las versiones del proyecto desarrollado.

Este tipo de herramientas están usualmente pensadas para desarrollo de aplicaciones en equipos de trabajo. Aun así tiene una serie de características que las hacen interesante para este proyecto:

- Copia de seguridad y recuperación
- Sincronización
- Deshacer cambios a corto plazo
- Deshacer cambios a largo plazo
- Seguimiento de cambios
- Seguimiento de la autoría

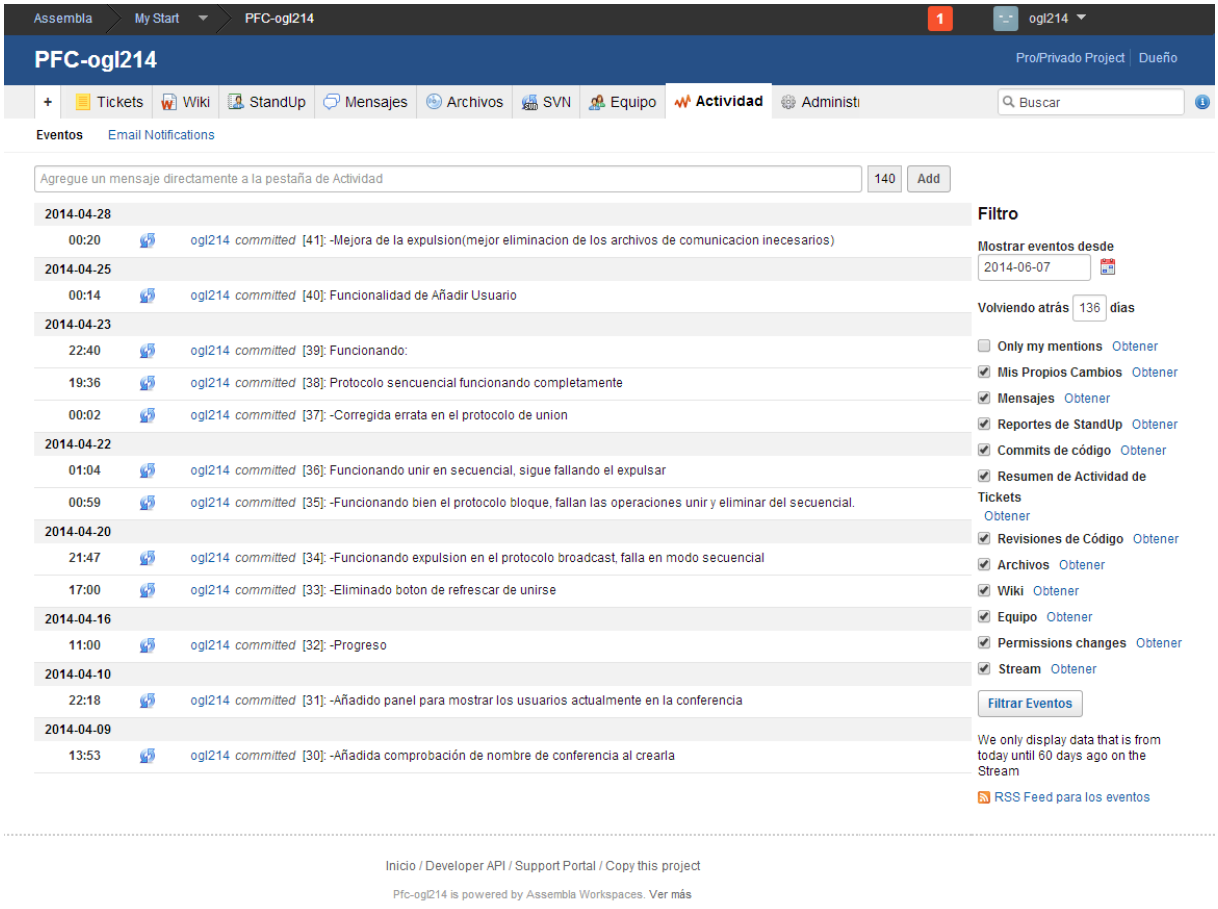

Figura 3.1: Assembla - Una herramienta de control de versiones

Una alternativa muy conocida a Subversion es GIT, que es utilizada por el popular sitio Web *GitHub*. En subversion todas las operaciones son realizadas directamente sobre el repositorio principal. En contraposición, en GIT se utiliza un repositorio local intermedio donde cada usuario puede utilizar todas las características y beneficios del control del código fuente sin que estos cambios sean reflejados en el servidor principal hasta que este usuario así lo deseé.

Ambas alternativas tienen sus ventanas e inconvenientes. Con GIT se garantiza que en el servidor central la versión del software siempre sea funcional dejando las funcionalidades a medio desarrollar en el servidor local de cada usuario. Esto no significa que el historial de cambios realizado por el usuario en local se pierda y solo quede constancia del último, pues todo el

historial de cambios realizados en local será comunicado al servidor central cuando se haga el *commit* a éste.

Por otro lado con Subversion todos los cambios *commiteados* realizados se reflejarán directamente en el servidor central. Pero con está alternativa se pierde el riesgo de poder perder la copia local si el equipo se daña antes de realizar el *commit* al repositorio central. Aunque esto también se podría realizar haciendo uso de GIT aunque se desaprovecharía la existencia del servidor local intermedio.

En este proyecto finalmente se ha optado por utilizar subversion(SVN), Figura 3.1, por su madurez, simplicidad y que tiene una mejor integración con Eclipse.

3.1.4. Dropbox API

En cuanto al uso de Dropbox, como ya se mencionó en la introducción, será un servidor central usado para realizar todo el intercambio de mensajes entre los usuarios. En este punto se verán las ventajas y desventajas de usar la API de DropBox en este proyecto, así como se compara con otras APIs.

Dropbox (http://www.dropbox.com) es un servicio de alojamiento de archivos multiplataforma en la nube, operado por la compañía Dropbox. El servicio permite a los usuarios almacenar y mantener los archivos sincronizados en linea, ya sea entre ordenadores o entre diferentes usuarios en forma de carpeta compartida.

Pero los servicios de DropBox no solo se limitan de cara al usuario estándar, sino que cuenta con una serie de servicios enfocados para desarrolladores. Un ejemplo es la DropBox Core API que proporciona un medio que permite leer y escribir ficheros en DropBox desde una aplicación. Esto incluye también soporte para funciones avanzadas como búsqueda, control de versiones y recuperar archivos. Ésta API encaja con aplicaciones basadas que necesitan hacer uso de un servidor.

Existen alternativas a la API de DropBox ofrecidas por otras compañías como son: Google Drive, Box, Skydrive o Mega. Ésta ultima, Mega, carece de una API para Java. En cuanto al resto —Skydrive, Google Drive y Box— cuentan con API para Java.

Como se puede ver en la tabla 3.1, OneDrive no cuenta con API para Java por lo que quedara descartado. En cuanto a Box se descarta debido a la estricta limitación de tamaño de los ficheros en su versión gratuita. En cuanto los restantes - Dropbox y Google Drive - resulta difícil decantarse en uno sobre otro, siendo ambos buenas opciones para la implementación de este proyecto, pero debido a que uno de los motivos por el que se eligió el lenguaje Java es por su facilidad de migración a Android y teniendo en cuenta ,que como se ve en la tabla, actualmente Google Drive no cuenta con SDK para Android, la mejor opción sera utilizar la API Dropbox.

El primer inconveniente que surge ante este modelo es que dependemos de un tercero. Si DrobBox deja de funcionar, la aplicación quedara inservible. Ya sea por una caída temporal del servicio o por el cierre del mismo. Mas este es siempre uno de los riesgos inherentes a delegar una tarea en un tercero. Pero Dropbox es un servicio afianzado a lo largo de varios años con una gran base de usuarios y desarrolladores que usan sus servicios y, actualmente, resulta confiable.

En cuanto a la privacidad, resulta también un tema delicado pues nuestros archivos pasaran por un tercero y no se puede saber a ciencia cierta que uso hará él con estos. Pero en esta

implementación esta problemática no supone un problema por dos motivos: primero, todos los documentos de los usuarios subidos a la aplicación estarán cifrados en DropBox; segundo, la clave secreta con la que se cifraran los archivos de la conferencia no puede ser obtenida a partir de la información intercambiada entre los usuarios sobre DropBox.

Servicio / Característica	OneDrive	Dropbox	Google Drive	Box
Tamaño máximo de los archivos	2GB	No para aplicaciones	10GB	250MB con el plan gratis, 5GB con el de pago
Almacenamiento gratuito	7GB	2GB	15GB	10GB
Planes de pago	25 dolares al año por cada 50GB (máximo 200GB)	10 dolares al mes por cada 100GB (máximo 500GB)	2 dolares al mes por 100GB o 10 dolares al mes por un 1TB	10 dolares por cada 100GB
SDKs	Widndows Phone, .Net, Android y iOS	Python, Ruby, PHP, Java, Android, iOS, OS X y HTTP	.NET, Java, JavaScript, Objetive-C, PHP y Phython	iOS, Android, Windows y Java
SDKs desarrollados por third-parties		C++, C#, JavaScript, Perl y Racket		C#, Ruby y Python

Tabla 3.1: Comparativa entre servicios de alojamiento de archivos

3.2. Diseño

Tras haber hecho una revisión de las tecnologías que se van a usar, se presenta el diseño de la solución. En primera instancia abordaremos el primer paso con diagramas UML.

3.2.1. UML

En este punto se van a ver los diagramas UML de la aplicación. Debido a las numerosas relaciones y extensión de un diagrama UML general, se va a proceder a verlo por secciones.

UML del protocolo

Éste diagrama UML,figura 3.2, se enfoca en la parte pertinente a la implementación del protocolo. Las estructuras matemáticas, las operaciones, encapsulado de los protocolos, etc.

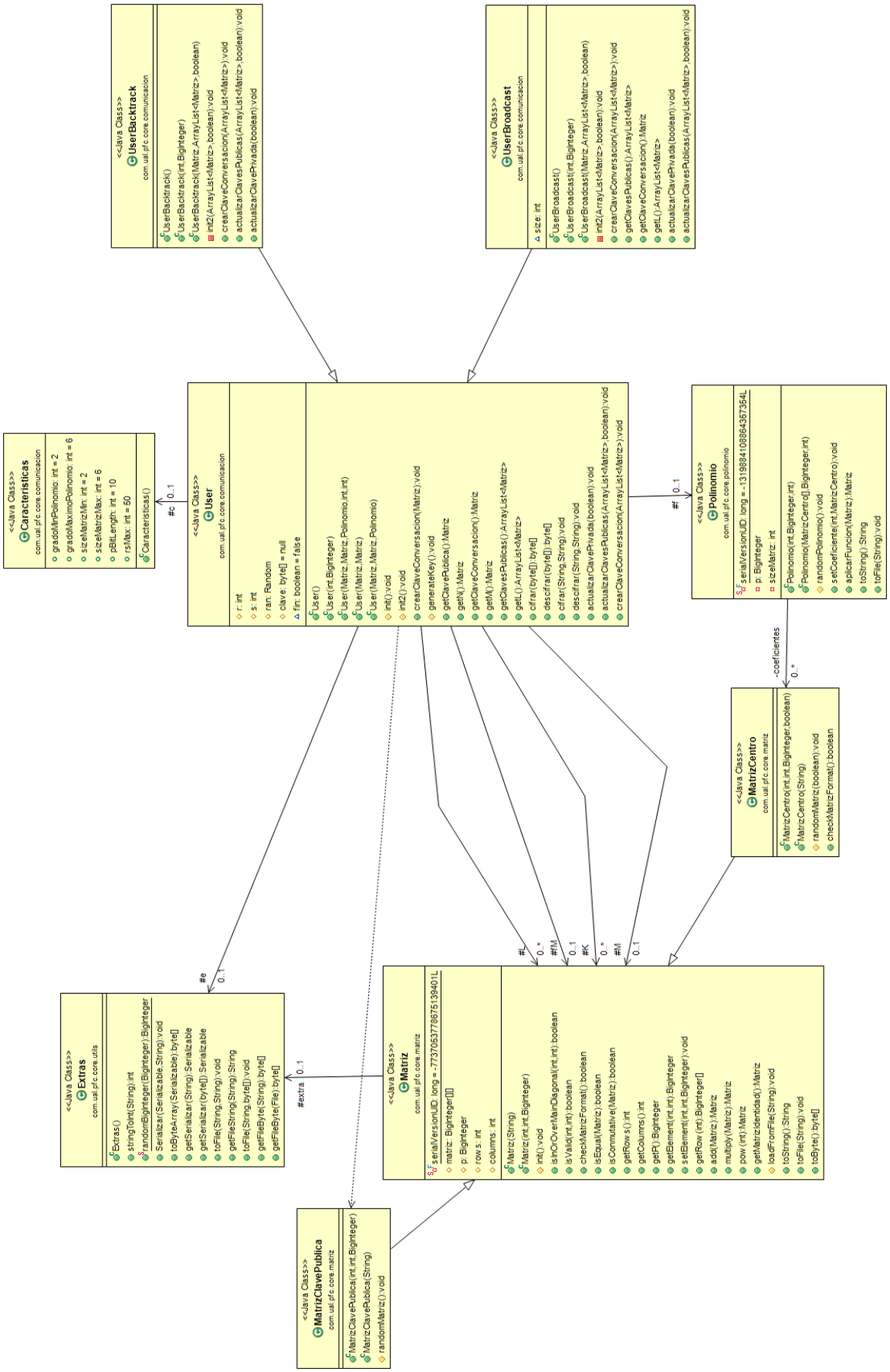

Figura 3.2: Diagrama UML de la implementación del protocolo

UML de la API de DropBox

Las clases representadas en siguiente diagrama UML, Figura 3.3, contiene principalmente dos elementos: la clase dropboxApi, que consiste en un intermediario con la API de Dropbox que supone una simplificación de ésta; la clase de threadUpdateDropbox, que será explicada con más detalle en el siguiente punto, consiste en un hilo que se ejecuta en segundo plano que lanza un evento — que escucharan todas las clases que se registren en el listener — cuando el archivo, o algún archivo de dentro de una carpeta, cambie.

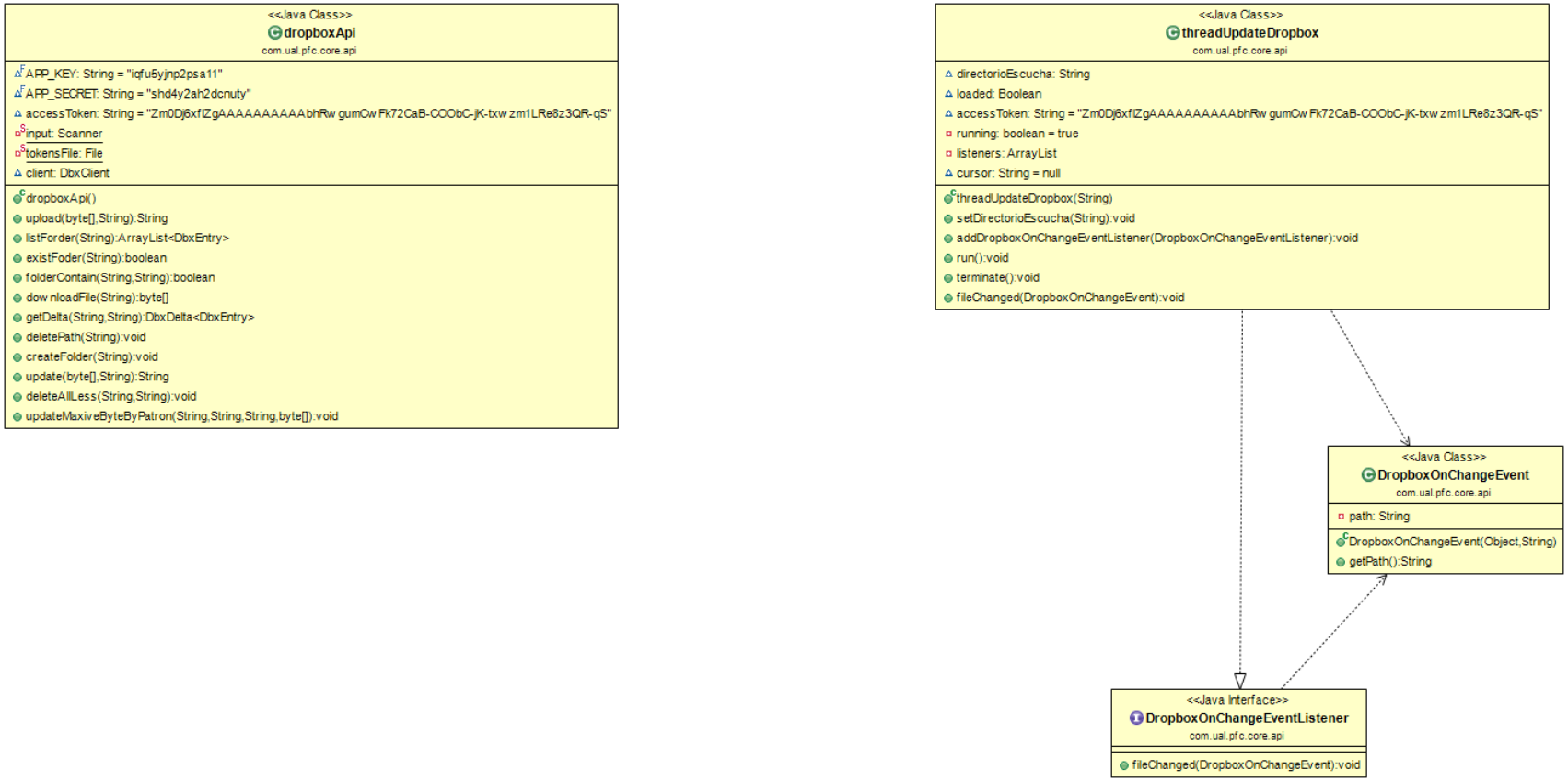

Figura 3.3: Diagrama UML de la comunicación con DropBox.

UML del esquema de datos

Estas clases, Figura 3.4, son usadas para gestionar la información de los usuarios, tales como nombre de usuario, contraseña, conferencias a las que esta invitado, etc. Hemos de destacar que UserList es serializable, y por lo tanto también UserInfo y Conferencia, con el fin de poder almacenar esta clase en DropBox, donde sera accesible por todos los usuarios.

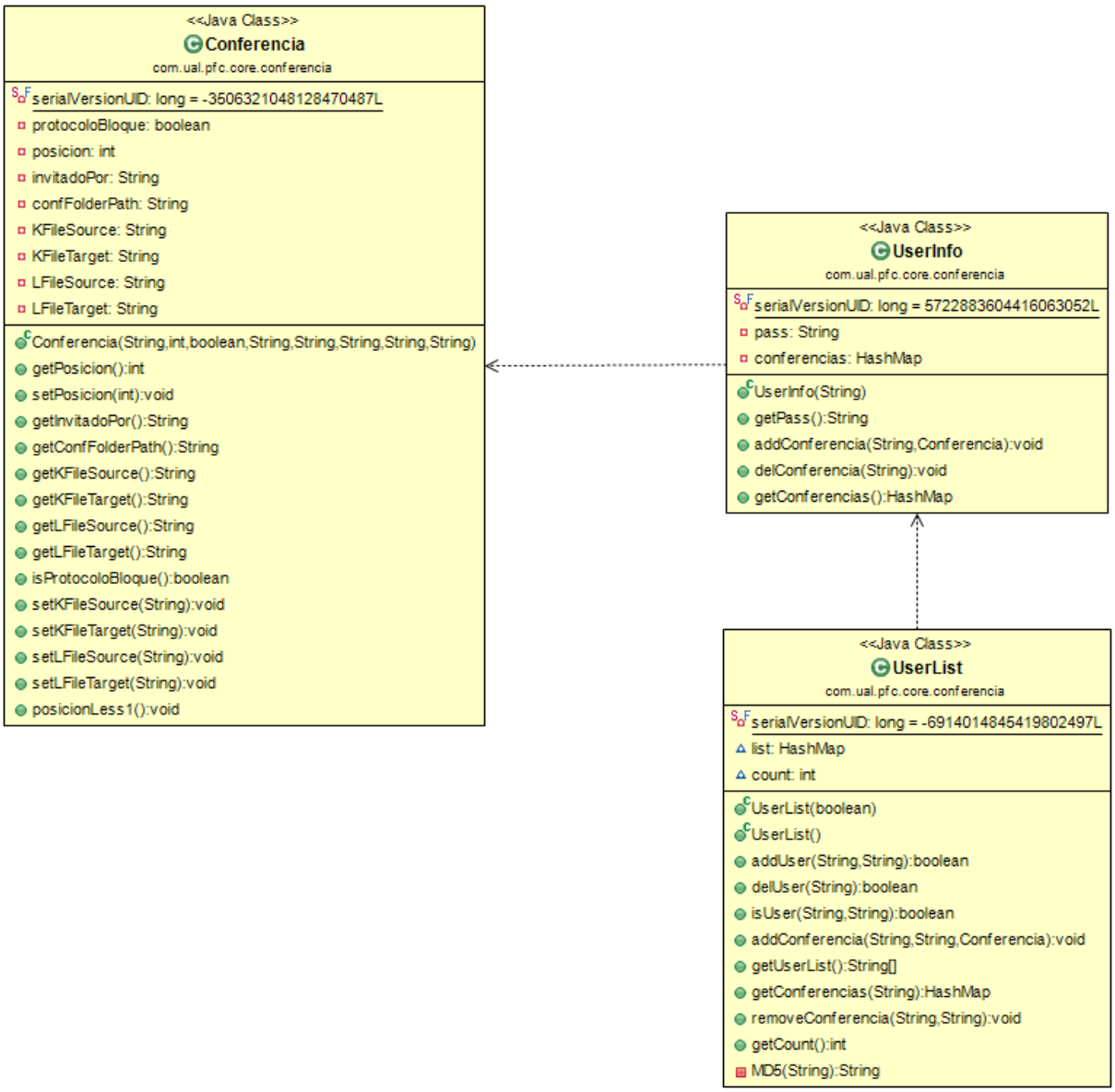

Figura 3.4: Diagrama UML del esquema de datos.

UML de la interfaz gráfica de usuario

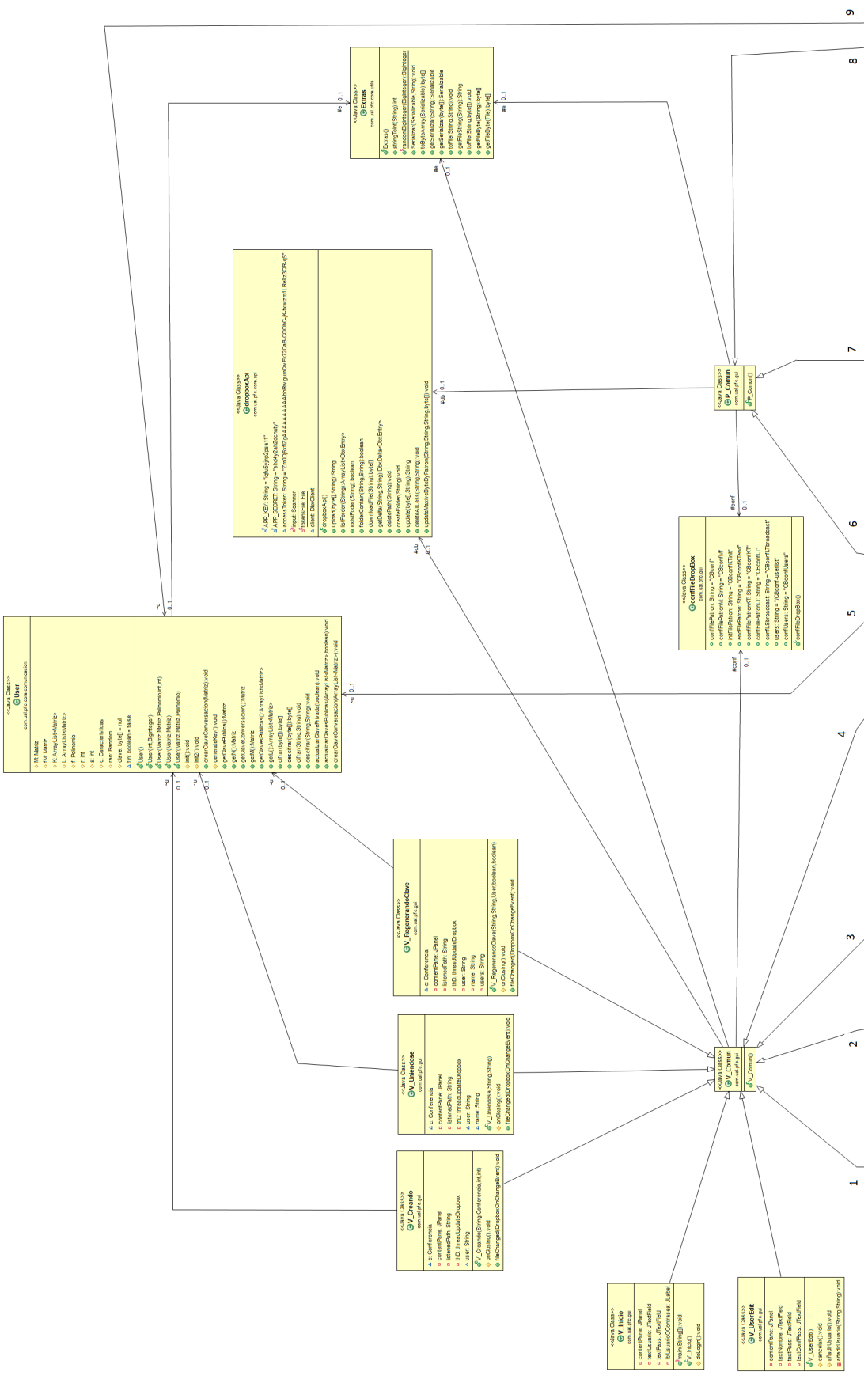

Figura 3.5: Diagrama UML de la interfaz gráfica de usuario. Parte I

Figura 3.6: Diagrama UML de la interfaz gráfica de usuario. Parte II

UML de la interfaz gráfica de usuario 2

En este diagrama, Figura 3.7,se dibujan las relaciones entre los elementos gráficos que harán uso del elemento threadUpdateDropbox y éste. Esto ha sido separado en otro diagrama para mantener una mayor claridad y simplicidad, y por tanto que sean más entendibles, en los diagramas.

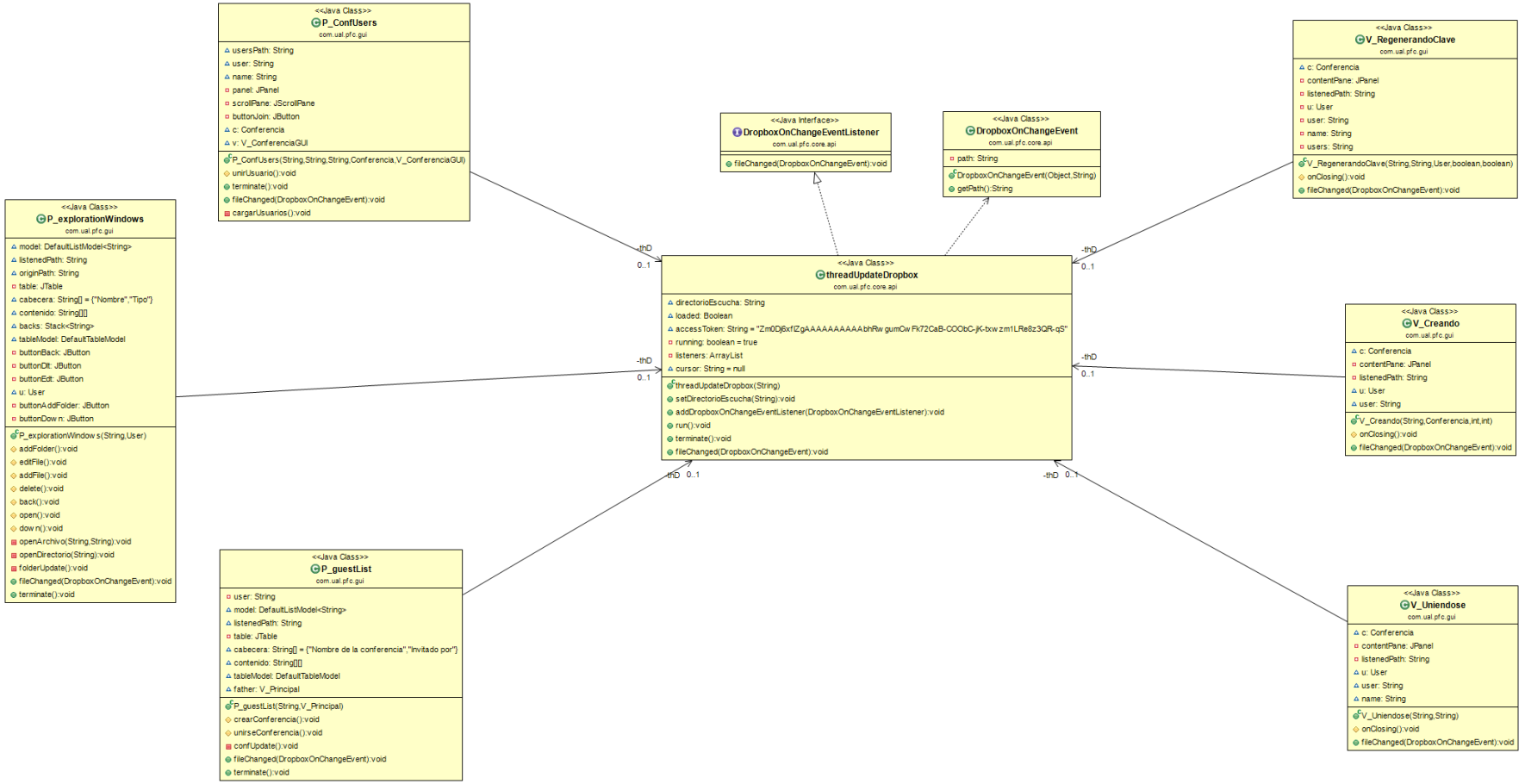

Figura 3.7: Diagrama UML de la interfaz gráfica de usuario 2.

3.2.2. Diagrama de estados

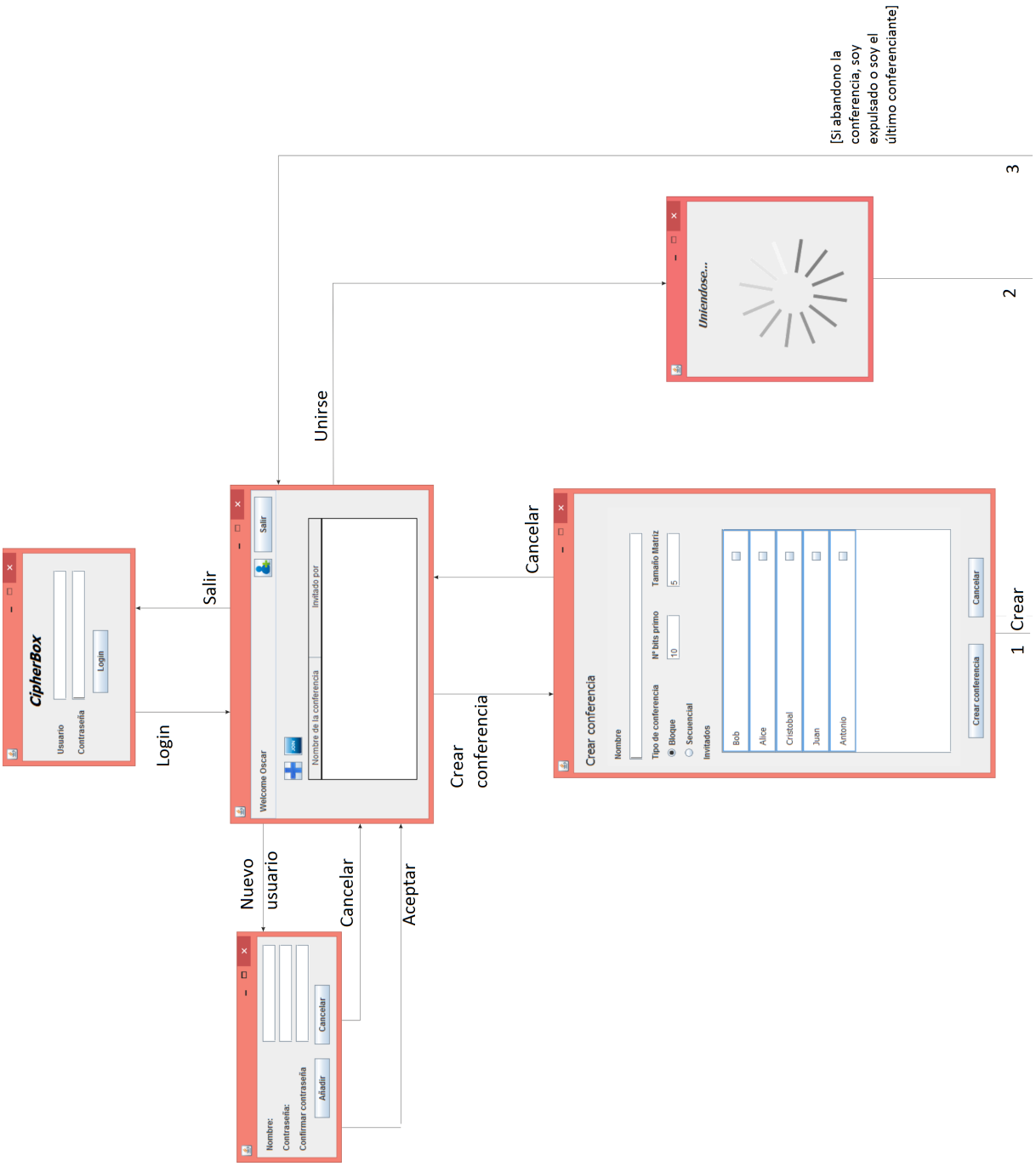

Figura 3.8: Diagrama de estados. Parte I

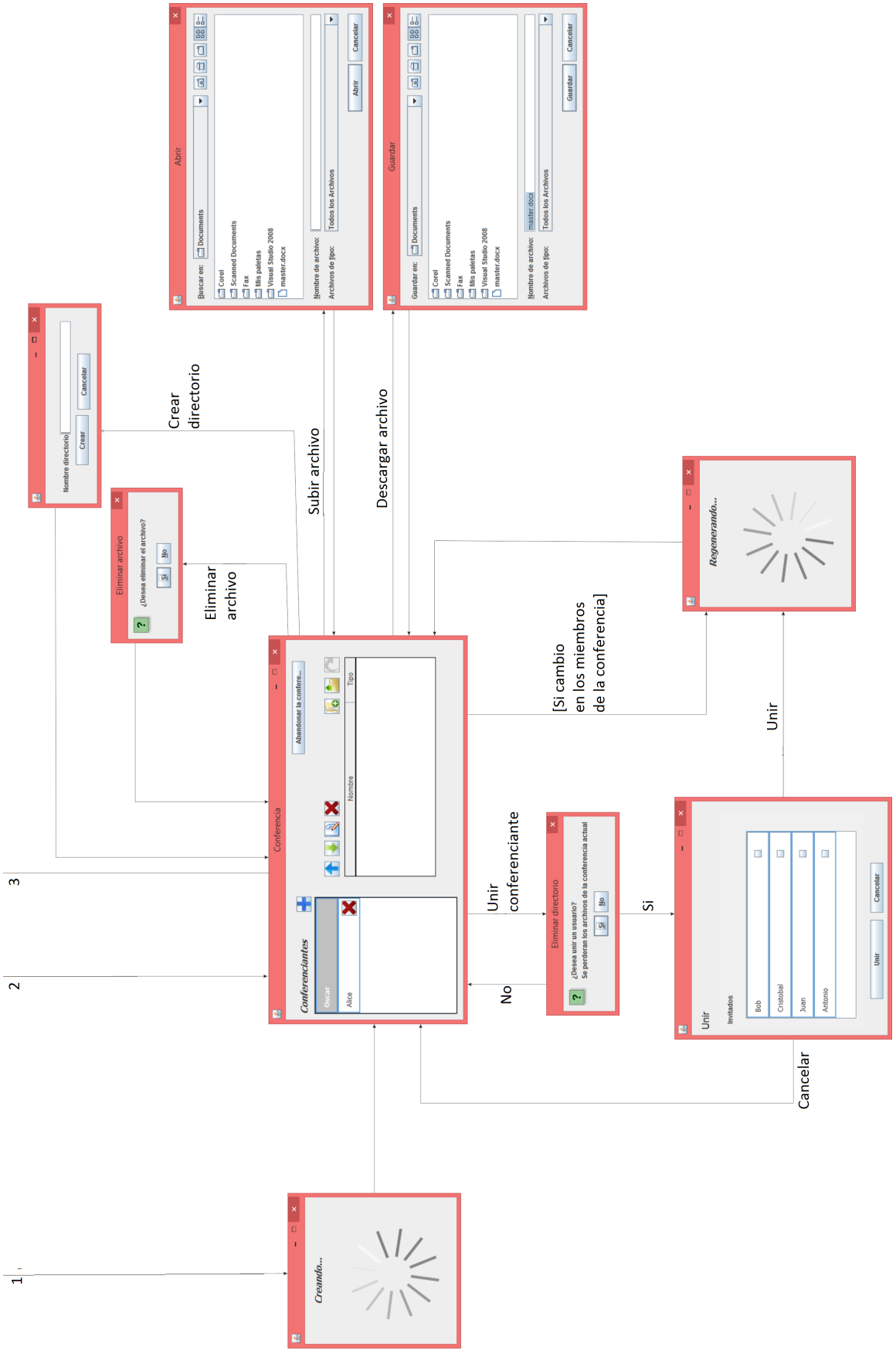

Figura 3.9: Diagrama de estados. Parte II

3.3. Implementación

En éste último punto del capítulo se tratan algunos aspectos relevantes relativos a la implementación.

3.3.1. DropBoxThread

La DropBox Core API carece de un sistema de eventos que avise a la aplicación cuando un archivo es modificado. Esto supone una carencia debido a la necesidad de esta aplicación de reaccionar cuando los usuarios envían sus mensajes a través de DropBox. Esto se encuentra solventado en la API DataStore, también de DropBox, que proporciona una mejor sincronización entre los dispositivos, almacenar también estructuras de datos, acceso offline y resolución automática de conflictos.

La solución a esta problemática sin utilizar la API DataStore, lo cual supone un trabajo futuro de este proyecto, pasa por el uso de un elemento de la API de Dropbox llamado: Dropbox's Delta. Éste permite saber todo el historial de modificaciones por el que ha pasado un archivo o carpeta. Por lo tanto, si el Delta de un fichero cambia entre dos consultas separadas temporalmente implicará que el fichero ha cambiado.

La solución propuesta consiste en un hilo que se ejecuta en segundo plano que irá comprobando el Delta del archivo o carpeta a *"monitorizar"* lanzando un evento a aquellas clases registradas en su listener cuando esto suceda.

Código 3.1: threadUpdateDropbox.java

```
1
2  public class threadUpdateDropbox extends Thread implements DropboxOnChangeEventListener{
3      ...
4
5      public void addDropboxOnChangeEventListener (DropboxOnChangeEventListener listener)
6      {
7          // agregamos el listener a nuestra lista
8          listeners.add(listener);
9      }
10
11     ...
12
13     public void run(){
14         DbxRequestConfig config = new DbxRequestConfig(
15                 "PathPrefixBlogPost/1.0", Locale.getDefault().toString());
16         DbxClient client = new DbxClient(config, accessToken);
17
18         while (running) {
19                 DbxDelta<DbxEntry> result = null;
20                 try {
21                     result = client.getDeltaWithPathPrefix(cursor, directorioEscucha);
22                     //directorioEscucha es un string con el nombre del path desde el que queremos
                           que se lance un evento cuando cambie
23                 } catch (DbxException e) {
24                     cursor=null;
```

```
25                      continue;
26                  }
27                  cursor = result.cursor;
28                  if(loaded==true)
29                  {
30                      //se lanza el evento a los listeners
31                      if (result.entries.size()!=0) {
32                          ListIterator li = listeners.listIterator();
33                          while (li.hasNext()) {
34                      ((DropboxOnChangeEventListener)li.next()).fileChanged(new
                            DropboxOnChangeEvent (this,directorioEscucha));
35                          }
36                      }
37                  }
38                  else{
39                      loaded=true;
40                  }
41
42                  if (!result.hasMore) {
43                      try {
44                          Thread.sleep(2000);
45                      } catch (InterruptedException e) {
46                          e.printStackTrace();
47                      }
48                   }
49              }
50      }
51
52      ...
53
54  }
```

En cada clase que quiera hacer usó de este hilo se deberá:

Código 3.2: Ejemplo de uso de threadUpdateDropbox

```
1   public class claseEscuchaEjemplo implements DropboxOnChangeEventListener {
2
3       ...
4
5       private threadUpdateDropbox thD; // declarar el hilo
6
7       ...
8
9       public claseEscuchaEjemplo{
10          ...
11          thD = new threadUpdateDropbox(listenedPath);//incializar el hilo.
12          //cuando ocurra algun cambio en listenedPath se lanzara un evento
13          thD.start();//lanzamos el hilo
14          thD.addDropboxOnChangeEventListener(this);//registramos esta clase en los listener del hilo
                para poder escuchar los eventos que lance
15          ...
16      }
17
18      ...
```

```
19
20      @Override
21      public void fileChanged(DropboxOnChangeEvent args) {
22          if(args.getPath().compareTo(usersPath)!=0){
23              return;
24          }
25          //Do things
26          ...
27      }
28
29      ...
30
31  }
```

3.3.2. Integridad de UserList en DropBox

En este apartado tratamos UserList, Figura 3.4 . UserList contendrá la información de los usuarios — nombre de usuario, contraseña, información de las conferencias a la que esta invitado, etc —. Éste se encontrará serializado en DropBox para que todos los usuarios puedan acceder a él e interactuar con el mismo. En el caso de que sea modificado en local, se deberá actualizar la versión existente con la actual.

Este archivo es susceptible de ser modificado por varios usuarios a la vez lo cual podría provocar conflictos. Estos conflictos causarán que la información pierda su integridad provocando un mal funcionamiento de la aplicación.

Pero usando la API de DropBox, y manteniendo un buen manejo de los casos de excepción, se puede garantizar que esto no suceda. Cada vez que se sube un archivo a DropBox se le indica la dirección a donde debe subirlo y devuelve la posición real a la que lo ha subido. En situaciones normales estas dos direcciones son la misma, pero si ocurre un conflicto diferirán.

Un ejemplo de como debe realizarse las interacciones con éste es el siguiente:

Código 3.3: Ejemplo de interacción con UserList

```
1  ...
2
3  do{
4      UserList ul= (UserList) e.getSerializar( db.downloadFile(conf.users));
5
6      ...//se opera sobre UserList
7
8      aux="/"+db.update(e.toByteArray(ul), conf.users);
9
10     if(aux.compareTo(conf.users)!=0)//comprobamos si ha sucedido algun conflicto
11     {
12         db.deletePath(aux);
13     }
14 }while(aux.compareTo(conf.users)!=0);//en el caso que haya habido un conflicto repetimos
15
16 ...
```

De esta manera se solventan los conflictos que puedan surgir manteniendo así la integridad de la información.

3.3.3. Intercambio de los mensajes sobre DropBox

El intercambio de claves se realizará usando la API de DropBox. La API de DropBox proporciona una carpeta dentro de la cual una aplicación puede operar. Subir, descargar y borrar archivos, crear nuevos directorios, ver el historial de cambios de estos, etc. Esta carpeta sera el medio mediante el cual se realizara el intercambio de mensajes, pero para que esto sea posible será necesario definir antes ciertas estructuras y metodologías que lo harán posible.

Todos los archivos usados internamente comenzaran por *"CBconf"*. Siendo ésta una palabra reservada que no podrá usarse en los archivos subidos por los usuarios.

El primer elemento será el archivo *"CBconf-userlist"*, Figura 3.4, situado en la raíz de la capeta. Éste será imprescindible y su propósito es mantener la gestión de los usuarios.

Contendrá la lista de todos los usuarios de confianza que pueden crear y participar en conferencias. De cada usuario guardará el nombre de usuario y su contraseña, ademas de la lista de conferencias a la que cada usuario esta invitado.

Lo siguiente será profundizar en el propósito de la clase conferencia. Al crear una conferencia con n usuarios se crearán n elementos del tipo de conferencia, uno para cada usuario. Cada uno de los elementos será diferente y entre todos harán posible el intercambio de claves.

El primero de los datos que contiene el elemento conferencia,Figura 3.10, es el nombre de ésta. Una conferencia será representada en DropBox mediante una carpeta, por tanto, no podrán existir a la vez dos conferencias con el mismo nombre. También se indica el protocolo de la conferencia, la posición que ocupa el conferenciante dentro de la conferencia y por quien ha sido invitado. El resto de campos están enfocados al intercambio de mensajes entre los usuarios y son;

- **KFileSource:** indica el archivo desde donde debe leer el vector de K elementos enviado por usuario anterior. En caso que sea el primer usuario contendrá *"CBconfKTinit"*.
- **KFileTarget:** indica el archivo donde debe escribir el vector de K elementos que leerá el siguiente usuario. En caso que sea el último usuario contendrá *"CBconfKTend"*.
- **LFileSource:** indica el archivo desde donde debe leer el vector de L elementos enviado por usuario anterior. En caso que se esté utilizando el protocolo en bloque será *"CBconfLT-broadcast"* para todos los conferenciantes.
- **LFileTarget:** indica el archivo donde debe escribir el vector de L elementos que leerá el siguiente usuario. En caso que se esté utilizando el protocolo en bloque será *"CBconfLT-broadcast"* para todos los conferenciantes.

Los archivos de intercambio de mensajes entre usuarios — a los que apuntan KFileSource, KFileTarget, etc — estarán dentro de la carpeta de la conferencia. La carpeta y estos serán creados por el usuario que cree la conferencia. El formato será la concatenación de *"CBconfKT"* — para el intercambio de los vectores de elementos K — o *"CBconfLT"* — para el intercambio de los vectores de elementos L — junto con dos números que representan la posición de los usuarios que se comunican mediante dicho archivo.

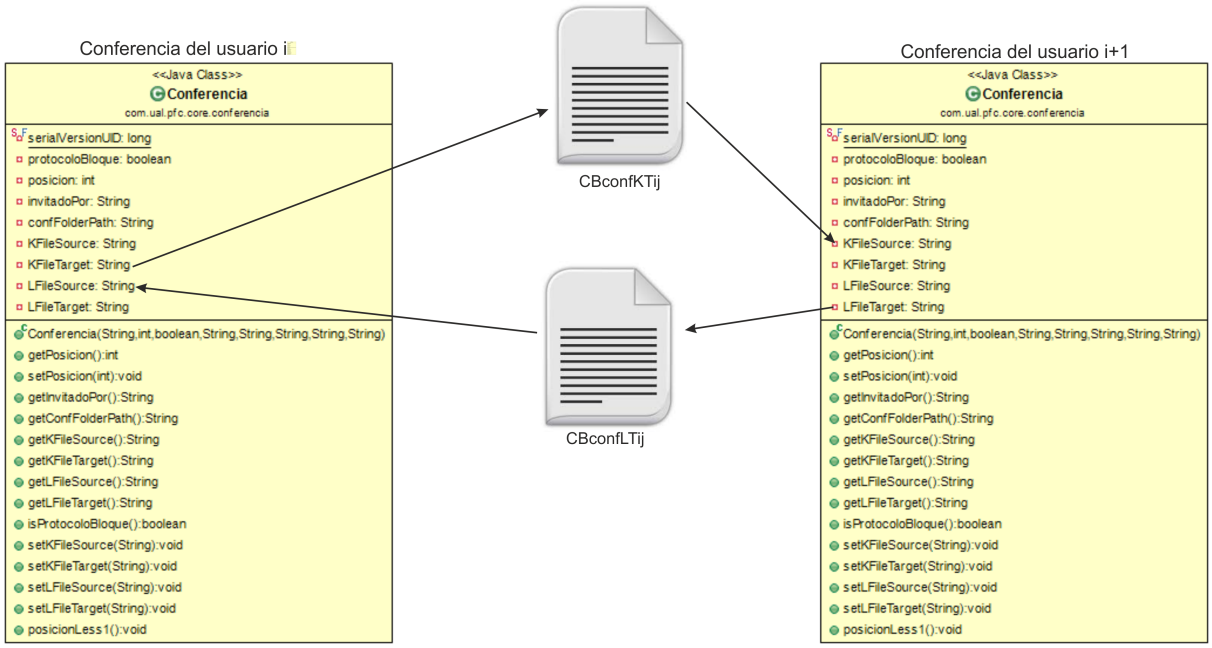

Figura 3.10: Esquema de intercambio de archivos entre dos usuarios

En el protocolo secuencial para n usuarios habrá 1 archivo *"CBconfM"*, que será la matriz M compartida por todos los usuarios, n-1 para el intercambio de los vectores de matrices K y otros n-1 para el intercambio de vectores de matrices n. En total 2n-1 archivos serán usados para el intercambio de información entre los usuarios.

En el protocolo en bloque para n usuarios habrá 1 archivo para la matriz M, n-1 para el intercambio de los vectores de matrices K y únicamente 1 para el intercambio de vectores de matrices n. En total n+1 archivos serán usados para el intercambio de información entre los usuarios.

Remarcar que ésta metodología garantizará el orden correcto de intercambio de mensajes entre los usuarios. Pues un usuario no escribirá el vector K o L, dependiendo de la etapa del protocolo en la que se encuentre, en su archivo destino hasta que el usuario anterior no escriba antes en el archivo origen de este usuario. En este proceso es donde resulta vital el uso de threadUpdateDropbox, introducido en el apartado anterior.

Dentro de la carpeta de la conferencia también habrá un archivo de configuración, *"CBconfUsers"*. Éste contendrá la lista de usuarios que participan en la conferencia, así como la posición que estos ocupan ,lo cual permitirá avisar y reaccionar cuando un usuario sea añadido o expulsado de la conferencia.

En el caso de que se añada un numero usuario se avisara al conferenciante que actualmente ocupa la última posición. Éste deberá cambiar su *KFileTarget* y *LFileSource*, de forma que coincidan con el *KFileSource* y *LFileTarget* del nuevo usuario.

Código 3.4: Unir conferenciante

```
1            ...
2            //se localiza al usuario que ocupa la ultima posicion
3            for(int k=0;k<ul.getUserList().length;k++){
```

```
4                   if(((Conferencia)ul.getConferencias(ul.getUserList()[k]).get(name)).getKFileTarget().
                        compareTo(conf.endFilePatron)==0 )
5                   {
6                       pos=k;
7                   }
8               }
9               //se obtiene el tipo de protocolo de la conferencia  protocoloBloque=((Conferencia)ul.
                    getConferencias(ul.getUserList()[pos]).get(name)).isProtocoloBloque();
10
11              //Se calcula los valores del archivo que se usaran para el intercambio de mensajes entre el
                    ultimo usuario y el nuevo usuario.
12              i=ul.getCount()-1;
13              j=i+1;
14              //se cambia el KFileTarget del ultimo usuario
15              ((Conferencia)ul.getConferencias(ul.getUserList()[pos]).get(name)).setKFileTarget(confPath
                    +"/"+conf.confFilePatronKT+i+""+j);
16              //auxArray[0] sera KFileSource del nuevo usuario que sera igual al KFileTarget del ultimo
                    usuario
17          auxArray[0]=((Conferencia)ul.getConferencias(ul.getUserList()[pos]).get(name)).getKFileTarget
                ();
18              db.upload(white, auxArray[0]);
19              //auxArray[1] sera KFileTarget del nuevo usuario
20              auxArray[1]=conf.endFilePatron;
21              //en caso del protocolo en bloque los LFile del nuevo usuario sera CBconfLTbroadcast, y
                    no sera necesario cambiar el LFile Source del ultimo usuario
22              if(protocoloBloque){
23                  auxArray[2]=confPath+"/"+conf.confLSbroadcast;
24                  auxArray[3]=confPath+"/"+conf.confLSbroadcast;
25              }
26              else{
27                  auxArray[2]=confPath+"/"+conf.confFilePatronLT+i+""+j;
28                  db.upload(white, auxArray[2]);
29
30                  //Se cambia el LFileSource del ultimo usuario
31                  ((Conferencia)ul.getConferencias(ul.getUserList()[pos]).get(name)).setLFileSource(
                        confPath+"/"+conf.confFilePatronLT+i+""+j);
32
33                  //El LFileTarget del ultimo usuario sera igual al LFileSource del ultimo usuario
34                  auxArray[3]=((Conferencia)ul.getConferencias(ul.getUserList()[pos]).get(name)).
                        getLFileSource();
35              }
36              ...
```

En el caso que un usuario sea expulsado, o abandone la conferencia, se localizará el usuario anterior a esté. O en el caso que el usuario expulsado sea el que ocupe la primera posición, el siguiente. En el primer caso, se pondrá al usuario anterior el *KFileTarget* y *LFileSource* del expulsado. En el segundo, se pondrá al usuario siguiente el *KFileSource* y *LFileTarget* del expulsado.

Código 3.5: Expulsar conferenciante

```
1           //Calculo la posicion del conferenciante del que necesito editar su informacion
2               int posToChange;
3               if(cExpulsado.getPosicion()==1){
```

```
4                   //en caso que el usuario expulsado o que abandona la conferencia sea el primero, sera
                        el siguiente
5                   posToChange=2;
6               }
7               else{
8                   //sino sera el anterior
9                   posToChange=cExpulsado.getPosicion()-1;
10              }
11              for(int i=0;i<ul.getUserList().length;i++ ){
12                  //en el caso que sea el usuario del que tenemos que editar su configuracion
13                  if(((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).getPosicion()==
                        posToChange){
14                      if(cExpulsado.getPosicion()==1){
15                          //si se expulso el primer usuario al usuario siguiente le ponemos el Kfilesource
                                y Lfiletarget del expulsado
16                          ((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).
                                setKFileSource( cExpulsado.getKFileSource() );
17                          ((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).setLFileTarget
                                ( cExpulsado.getLFileTarget());
18                          db.deletePath( cExpulsado.getKFileTarget());
19                          if(!cExpulsado.isProtocoloBloque())
20                          {
21                              db.deletePath( cExpulsado.getLFileSource());
22                          }
23                      }
24                      else{
25                          //si no es el usuario primero al usuario anterior le ponemos el Kfiletarget y
                                Lfilesource del expulsado
26                          ((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).
                                setKFileTarget( cExpulsado.getKFileTarget() );
27                          ((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).setLFileSource
                                ( cExpulsado.getLFileSource());
28                          db.deletePath( cExpulsado.getKFileSource());
29                          if(!cExpulsado.isProtocoloBloque())
30                          {
31                              db.deletePath( cExpulsado.getLFileTarget());
32                          }
33                      }
34                  }
35                  //Se reduce la posicion de los conferenciantes que estuvieran despues del expulsado
36                  if(((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).getPosicion()>
                        cExpulsado.getPosicion() ){
37                      ((Conferencia)ul.getConferencias(ul.getUserList()[i]).get(name)).posicionLess1();
38                  }
39              }
```

Por último, cuando la conferencia termine la carpeta que contiene toda la información de la conferencia será borrada. Eliminando así cualquier rastro de ésta.

3.3.4. Intercambio de mensajes

Este punto se enfoca en como se realiza el envió mensajes — subir el archivo serializado del vector de elementos K o L — o recibir mensajes — descargar el archivo —.

El punto anterior definió a donde debían subirse y de donde debían bajarse. En éste se verá el como.

Lo primero será definir que contendrán los archivos, o mensajes, que se subirán o descargaran. Estos serán una serialización de un elemento del tipo "Vector"que está compuesta por la clase Matriz. Es decir, *"Vector<Matriz>"*.

Para enviar un mensaje, se convertirá el ArrayList<Matriz>que representara el vector de matrices en en un conjunto de bits y será subido a DropBox como un archivo binario.

Código 3.6: Enviar mensajes a través de un fichero por DropBox

```
1        db.update(e.toByteArray(u.getClavesPublicas()), c.getKFileTarget()) ;
```

Para saber cuando el usuario anterior ya ha mandado el mensaje, y por tanto ya se puede descargar, se usara el hilo *threadUpdateDropbox*.

Código 3.7: Uso del hilo threadUpdateDropbox para esperar recepción de mensajes a través de un fichero

```
1        thD=new threadUpdateDropbox(listenedPath);
2        thD.start();
3        thD.addDropboxOnChangeEventListener(this);
4        setResizable(false);
5
6        //en el caso que el usuario anterior editara el fichero antes que se lance el hilo
7        if(new String("white").compareTo(new String(db.downloadFile(listenedPath)))!=0){
8            Thread t1 = new Thread(new Runnable() {
9                public void run()
10               {
11                   fileChanged(new DropboxOnChangeEvent(this, listenedPath));
12               }});
13           t1.start();
14       }
```

Cuando el hilo avisé de que el mensaje ya está. Se descargará el archivo binario de DropBox y se convertirán los bits de éste en un ArrayList<Matriz>.

Código 3.8: Obtener mensajes a través de un fichero desde DropBox

```
1        (ArrayList<Matriz>)e.getSerializar(db.downloadFile(c.getKFileSource()))
```

Para la interacción con la API DropBox se usa la clase intermediaria *"dropboxAPI.java"*. Con la que se facilita la iterración con la API de Dropbox. Puede verse con detalle en los apéndices. En los apéndices también se puede ver como convertir ArrayList<Matriz>en un conjunto de bits y viceversa.

3.3.5. Cifrado y descifrado usando JCA

El cifrado y descifrado se hará a nivel de bits usando la librería: Java Cryptography Architecture (JCA), que incluye gran variedad de funcionalidades para el trabajo criptográfico.

El cifrado que se aplicará sobre los archivos será AES(Advanced Encryption Standard). AES[7], también conocido como Rijndael. Es un algoritmo simétrico de cifrado por bloques de 128 bits mediante el uso de claves de 128, 192 o 256 bits.

AES fue anunciado por el Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos el 26 de noviembre de 2001 después de un proceso de estandarización que duró 5 años. Se transformó en un estándar efectivo el 26 de mayo de 2002. Y desde 2006, AES es uno de los algoritmos más populares usados en criptografía simétrica.

El cifrado y descifrado se realizarÁ cada vez que se suba y descargué un archivo de DropBox usando la aplicación. Recordemos que esto se realizara una vez los usuarios ya han decidido la clave secreta de la conferencia.

Código 3.9: Cifrar

```
1     public byte[] cifrar(byte[] value) {
2         byte[] encrypted = null;
3
4         SecretKeySpec skeySpec = new SecretKeySpec(clave, "AES");
5         try {
6             Cipher cipher = Cipher.getInstance("AES");
7
8             cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
9             encrypted = cipher.doFinal(value);
10        } catch (Exception e) {
11            e.printStackTrace();
12        }
13        return encrypted;
14    }
```

Código 3.10: Descifrar

```
1     public byte[] descifrar(byte[] encrypted) {
2         byte[] original = null;
3         try {
4             SecretKeySpec skeySpec = new SecretKeySpec(clave, "AES");
5             Cipher cipher = Cipher.getInstance("AES");
6             cipher.init(Cipher.DECRYPT_MODE,
7                     new SecretKeySpec(skeySpec.getEncoded(), "AES"));
8             original = cipher.doFinal(encrypted);
9
10            return original;
11        } catch (Exception e) {
12            e.printStackTrace();
13        }
14        return original;
15    }
```

3.3.6. Pruebas unitarias de los elementos matemáticos y de los protocolos

Para garantizar el correcto funcionamiento de todos los elementos matemáticos que intervienen en el protocolo, y también del protocolo en sus dos variantes, se han realizado pruebas unitarias sobre estos usando JUnit.

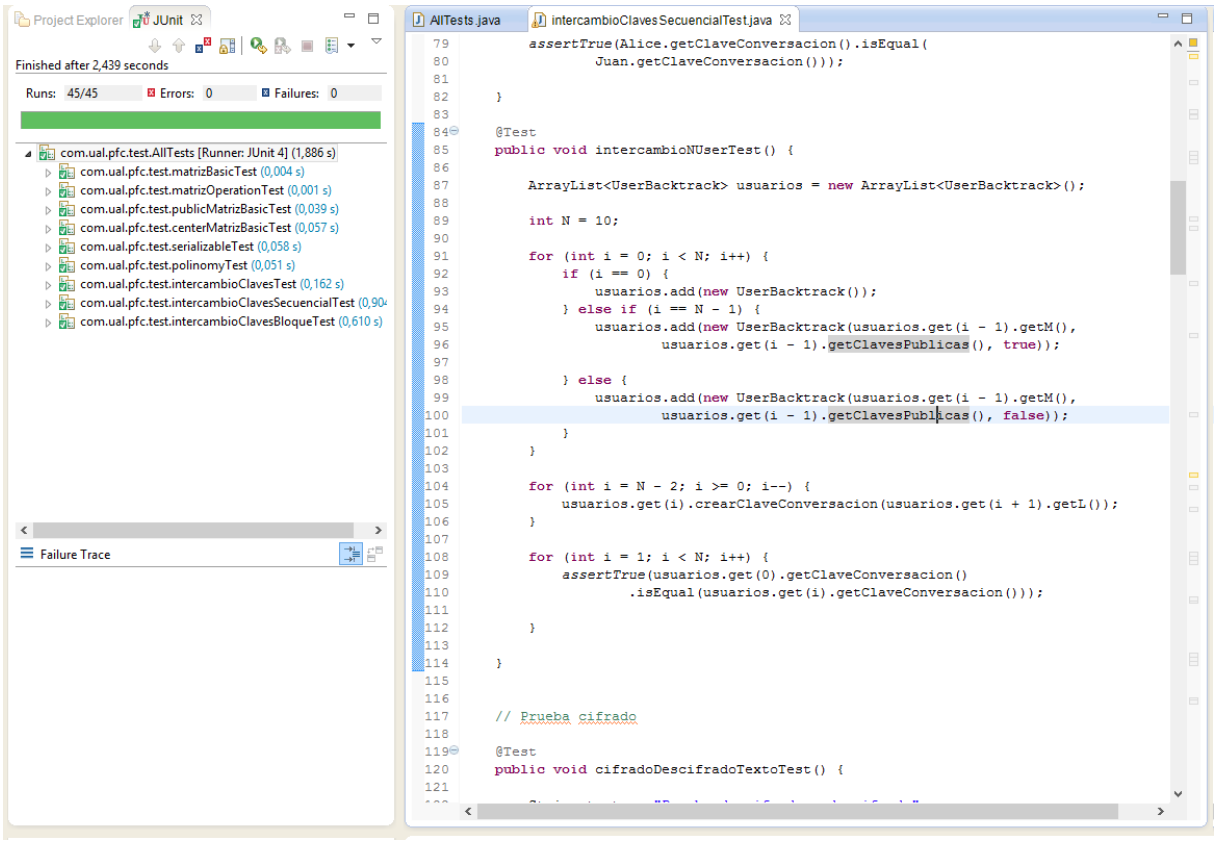

Figura 3.11: Test unitarios de los elementos del protocolo

En la parte matemática resulta primordial poder garantizar su correcto funcionamiento pues es uno los pilares del proyecto. Si los elementos matemáticos o sus operaciones fallaran, provocaría que el protocolo falle, eso llevaría a que los usuarios no tendrían la misma clave secreta y la comunicación entre ellos seria imposible.

Por ello se dedico especial atención al desarrollo de test para probar todos los posibles casos. En el apéndice pueden encontrarse algunos ejemplos de estos tests.

Otra ventaja de los test unitarios es que nos permite medir cuanto tiempo tarda cada protocolo bajo diferentes especificaciones en obtener la clave secreta compartida. Lo cual sera muy útil en el siguiente punto donde se realiza una comparativa entre ambos protocolos.

3.3.7. Comparativa de los protocolos

En este punto se va a proceder a hacer una comparativa de como evoluciona el tiempo según diferentes variables que afectan al protocolo. Los tiempos mostrados a continuación representan únicamente el tiempo invertido en realizar los cálculos matemáticas requeridos en la multidifusión de la clave.

Según el numero de bits del numero primo

Esta primera comparativa será del aumento del tiempo según crece el numero primo. En las Tablas 3.2, 3.3 y 3.4 puede verse esta evolución bajo diferente tamaño de la matriz y en la Gráfica 3.12 puede verse una comparativa de todas las tablas anteriores.

El tiempo crece en todas las tablas según aumenta el numero primo. También se puede observar que el tiempo en realizar los cálculos resulta más lento en el protocolo en bloque debido a la mayor cantidad de cálculos que este requiere — aunque esto se ve compensado debido a que requerirá de un menor trabajo de trabajo de red —.

Por otro lado, también se observa que el tiempo aumenta drásticamente según lo hace el tamaño de la matriz. Esto se debe a que los números con los que se trabaja crecen exponencialmente. Con una matriz de tres por tres y usando el numero primo 5, se manejaran números de hasta $5^3 = 125$. Con una matriz de cinco por cinto y el mismo primo, será $5^5 = 3125$. Y con una de diez por diez será $5^10 = 9765625$.

Por tanto, se puede ver que se deberá buscar un compromiso entre el numero primo y el tamaño de la matriz, pues si los dos aumentan considerablemente el tiempo invertido será desmesurado.

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	8	3	3	20	10	0,103543149	0,235349252
3	16	3	3	20	10	0,026438115	0,026994309
3	32	3	3	20	10	0,020230763	0,023952266
3	64	3	3	20	10	0,031050557	0,032707584
3	128	3	3	20	10	0,036589782	0,050615477
3	256	3	3	20	10	0,063778876	0,08092551
3	512	3	3	20	10	0,16877021	0,235349252
3	1024	3	3	20	10	0,625964948	0,763209723
3	2048	3	3	20	10	1,868387319	2,447329931
3	4096	3	3	20	10	4,571704757	8,341327951

Tabla 3.2: Comparativa según el numero de primo. Matriz de tamaño 3

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	8	5	6	20	10	0,047158825	0,023450809
3	16	5	6	20	10	0,018498747	0,022412531
3	32	5	6	20	10	0,02173676	0,028962271
3	64	5	6	20	10	0,027337312	0,03516154
3	128	5	6	20	10	0,054846257	0,073886921
3	256	5	6	20	10	0,150282841	0,206293938
3	512	5	6	20	10	0,576614471	0,782251001
3	1024	5	6	20	10	2,129232702	3,401674336
3	2048	5	6	20	10	10,90192733	22,66519851
3	4096	5	6	20	10	35,46513055	41,21293163

Tabla 3.3: Comparativa según el numero de primo. Matriz de tamaño 5

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	8	10	9	20	10	0,118402761	0,117783484
3	16	10	9	20	10	0,105171227	0,129680741
3	32	10	9	20	10	0,154860139	0,20705748
3	64	10	9	20	10	0,333126673	0,450274143
3	128	10	9	20	10	0,95817213	1,367166689
3	256	10	9	20	10	3,788517571	5,174181908
3	512	10	9	20	10	20,80317357	39,8626163
3	1024	10	9	20	10	77,79019604	100,104835
3	2048	10	9	20	10	157,6285995	211,6452832
3	4096	10	9	20	10	667,2125621	1027,005653

Tabla 3.4: Comparativa según el numero de bits del primo. Matriz de tamaño 10

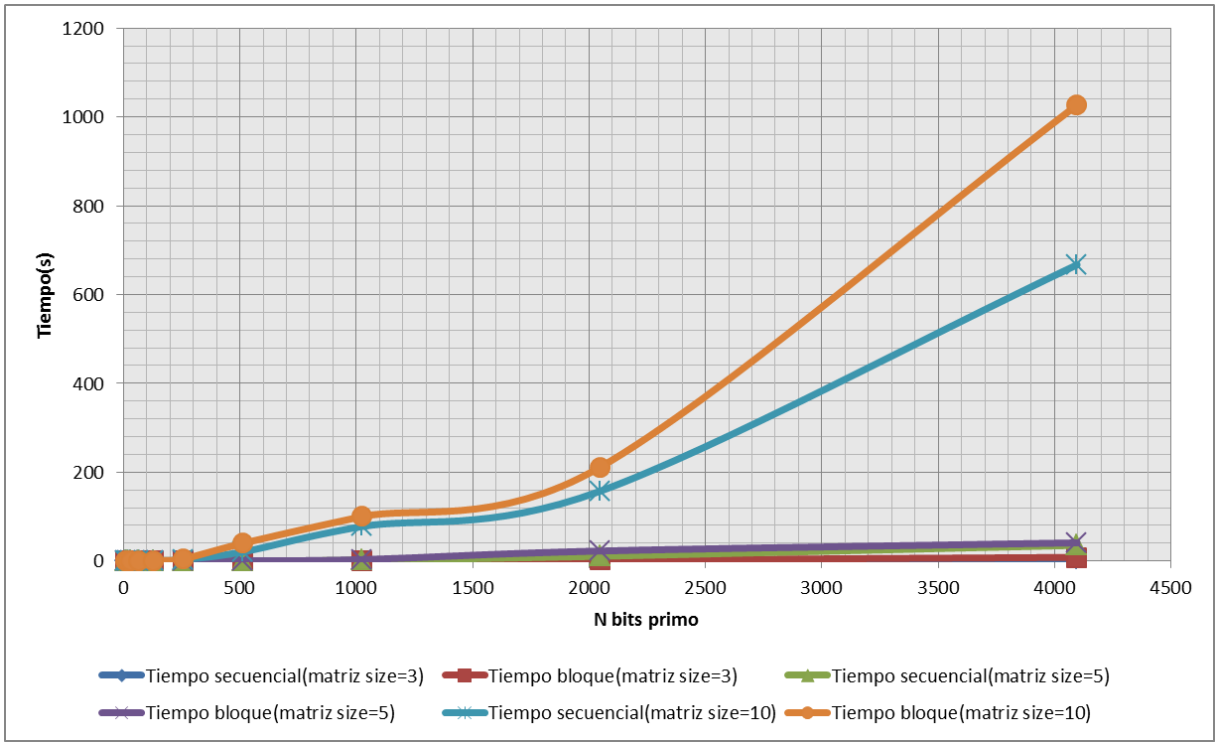

Figura 3.12: Comparativa según el numero de bits del primo

Según el tamaño de la matriz

Esta segunda comparativa se enfocara en la evolución del tiempo según tamaño de la matriz. En la anterior se realizó una comparativa donde influenciaba el tamaño de la matriz, mas esta estaba en un segundo plano enfocándose en el numero primo

En la Gráfica 3.13 y Tabla 3.5 se puede comprobar claramente que el tiempo aumenta exponencialmente.

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	16	3	3	20	10	0,023564104	0,012283048
3	16	5	3	20	10	0,017256723	0,015649248
3	16	7	3	20	10	0,0305936	0,041038586
3	16	9	3	20	10	0,051130552	0,07411944
3	16	15	3	20	10	0,27610884	0,39639289
3	16	30	3	20	10	4,917525792	11,9090493
3	16	50	3	20	10	52,58459439	96,75014222

Tabla 3.5: Comparativa según el tamaño de la matriz

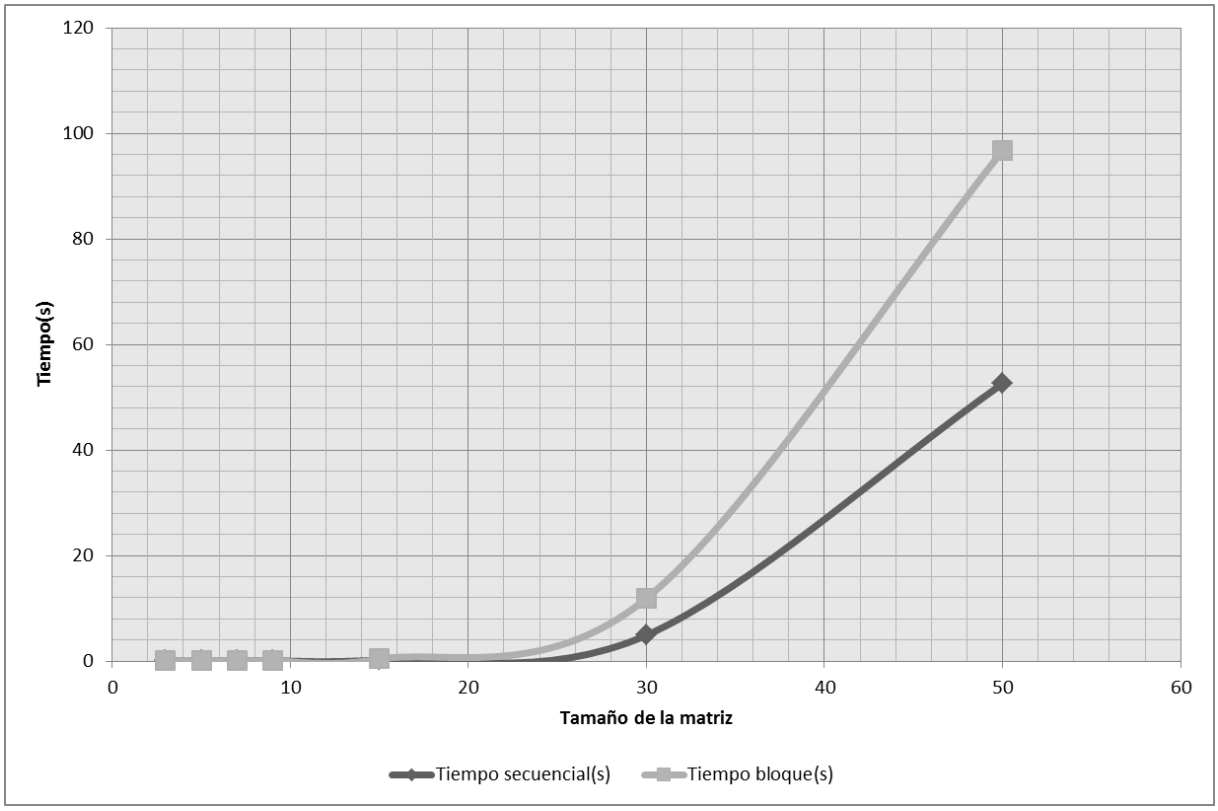

Figura 3.13: Comparativa según el tamaño de la matriz

Según el grado del polinomio

Ahora vamos a ver como afectan los elementos que componen la clave privada de los usuarios. En esta comparativa se verá como afecta uno de estos elementos, el grado del polinomio.

En los resultados, Gráfica 3.14 y Tabla 3.6, se ve que el grado del polinomio afecta en menor medida que el tamaño de la matriz y el primo, ya que éste provoca un aumento más lineal.

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	16	5	3	20	10	0,018185462	0,022570633
3	16	5	5	20	10	0,01583059	0,02002996
3	16	5	7	20	10	0,021485614	0,020016781
3	16	5	9	20	10	0,02082232	0,026842576
3	16	5	11	20	10	0,017166008	0,022444643
3	16	5	13	20	10	0,019942892	0,025450884
3	16	5	15	20	10	0,022089867	0,024929438
3	16	5	30	20	10	0,051599941	0,056422498
3	16	5	60	20	10	0,175804012	0,192635803
3	16	5	120	20	10	0,671037597	0,707596497

Tabla 3.6: Comparativa según el grado del polinomio

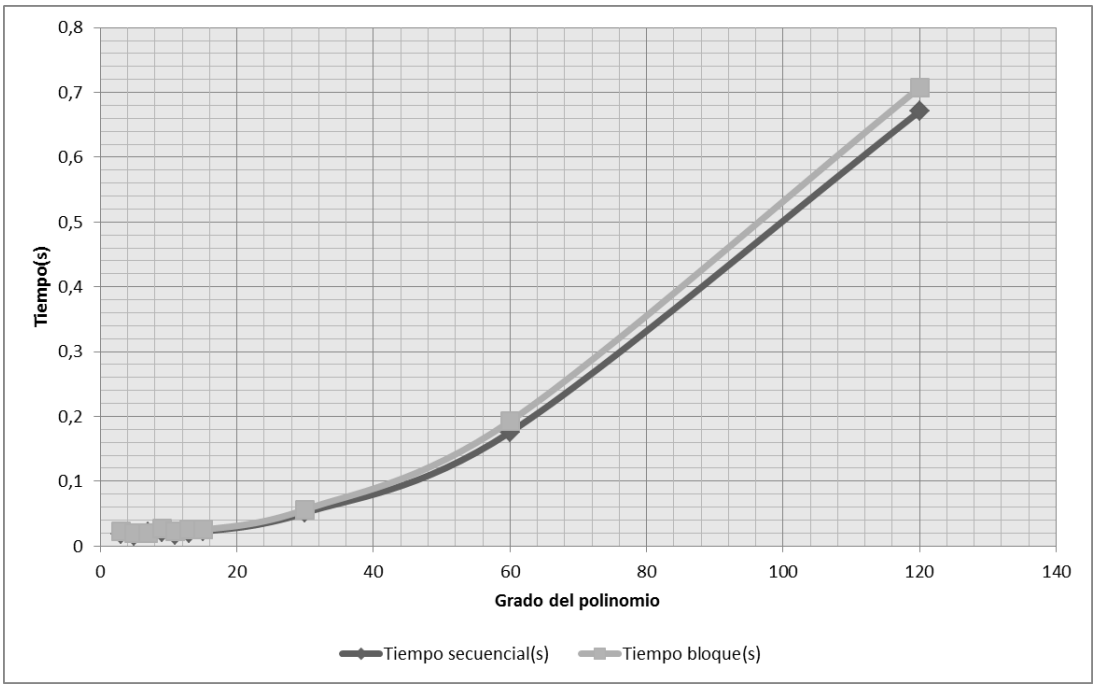

Figura 3.14: Comparativa según el grado del polinomio

Según los valores r y s

Esta comparativa sigue con los elementos que componen la clave privada de un usuario, los valores r y s. Como se puede ver en los resultados, Gráfica 3.15 y Tabla 3.7.

Éste será uno de los principales puntos a atacar si se quiere mejorar la velocidad del protocolo. R y s se usan como exponenciales a los que se elevara la matriz f(M). En esta implementación esta operación, la exponencial de una matriz, se ha realizado de forma secuencial, siendo por tanto mejorable si se realizara de forma paralela.

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial(s)	Tiempo bloque(s)
3	16	5	5	20	10	0,027530339	0,024917533
3	16	5	5	40	20	0,028296077	0,032842185
3	16	5	5	80	40	0,044371267	0,056098165
3	16	5	5	160	80	0,066263143	0,09648255
3	16	5	5	320	160	0,12498804	0,181652064
3	16	5	5	640	320	0,245936482	0,374331357

Tabla 3.7: Comparativa según los valores r y s

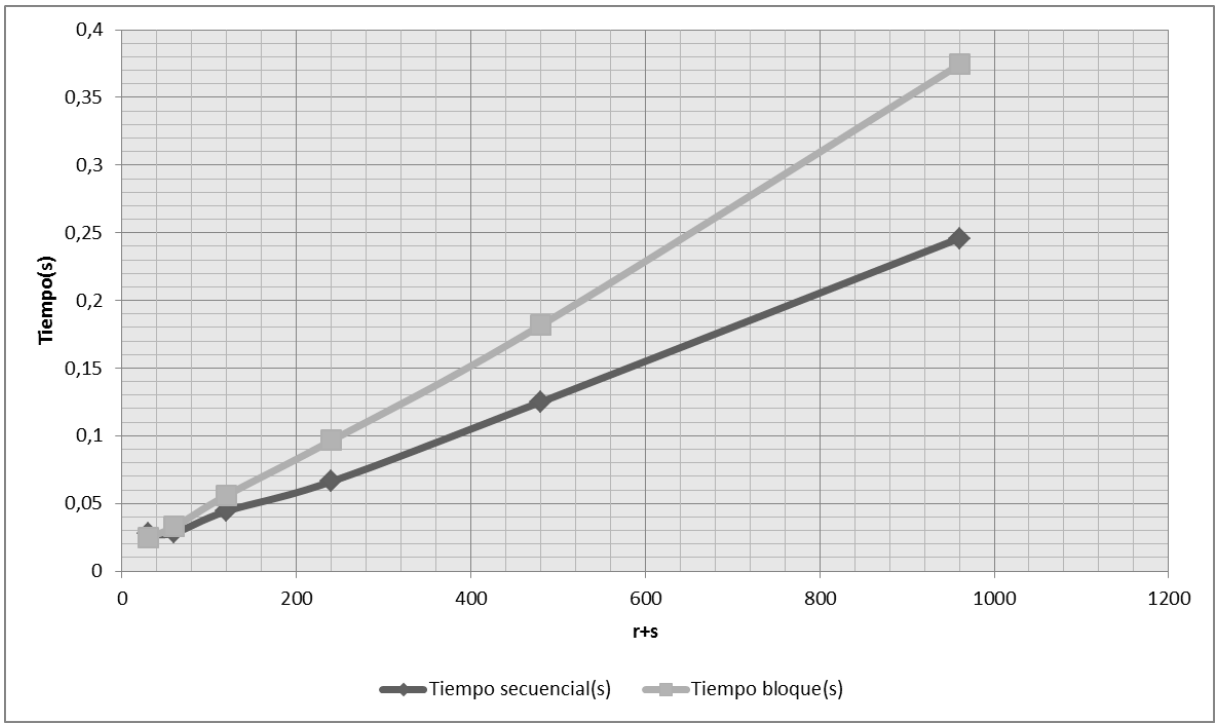

Figura 3.15: Comparativa según el grado del polinomio

Según el numero de conferenciantes

Por último en esta comparativa se vera la influencia del numero de usuarios que participan en la conferencia. Gráfica 3.16 y Tabla 3.8. Como se puede ver en estas — y en las anteriores comparativas — el protocolo secuencial resulta mas veloz que el protocolo en bloque.

Pero esta gráfica resulta engañosa, ya que solamente compara el tiempo invertido en realizar los cálculos matemáticos. Teniendo en cuenta que el protocolo secuencial requiere del doble de mensajes que el protocolo en bloque — y por tanto del doble de tiempo — será previsible que el protocolo en bloque tenga mejores resultados que el secuencial con grupos amplios de usuarios.

Como conclusión de esta comparativa. El protocolo secuencial será más aconsejable para conferencias con un numero pequeño de usuarios. Mientras que el bloque será aconsejable con grupos con un gran numero de usuarios.

Nº conferenciantes	Nº bits Primo	Tamaño Matriz	Grado Polinomio	r	s	Tiempo secuencial	Tiempo bloque
2	10	3	3	20	10	0,031326259	0,009890241
3	10	3	3	20	10	0,009532391	0,008517703
5	10	3	3	20	10	0,013385859	0,017081268
10	10	3	3	20	10	0,029842622	0,048038825
50	10	3	3	20	10	0,326621697	0,818618499
100	10	3	3	20	10	1,458283448	4,630591052
200	10	3	3	20	10	6,870110681	30,79330042
500	10	3	3	20	10	70,787087	118,87723

Tabla 3.8: Comparativa según el numero de conferenciantes

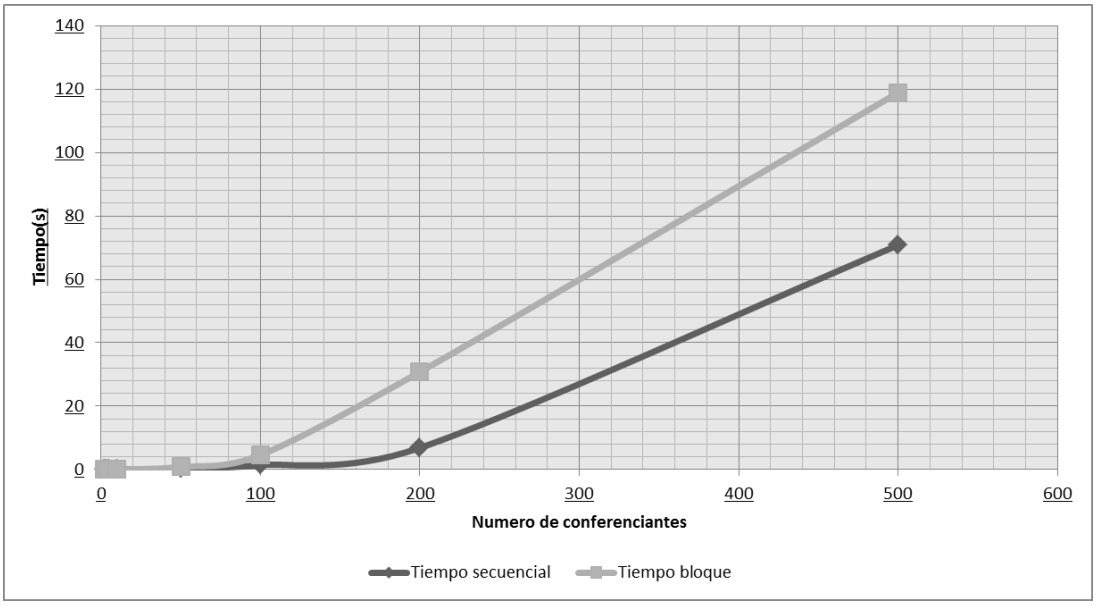

Figura 3.16: Comparativa según el numero de conferenciantes

Capítulo 4

Casos de uso

En este capitulo se probara el correcto funcionamiento de las funcionalidades de la aplicación. Para ello este capitulo se dividirá en diferentes secciones, cada una de ella representando una funcionalidad. Esto será especialmente útil por dos razones: poder plasmar en la documentación el correcto funcionamiento de la aplicación y servir de guía de uso de la misma.

4.1. Caso de uso 1: Crear conferencia

El primer caso de uso tratará sobre la funcionalidad más básica de la aplicación: crear una conferencia donde se puedan intercambiar ficheros de forma segura.

El primer paso será autentificarse en la aplicación. Destacar esta primera versión de la aplicación solo será posible registrarse mediante la invitación de otro usuario.

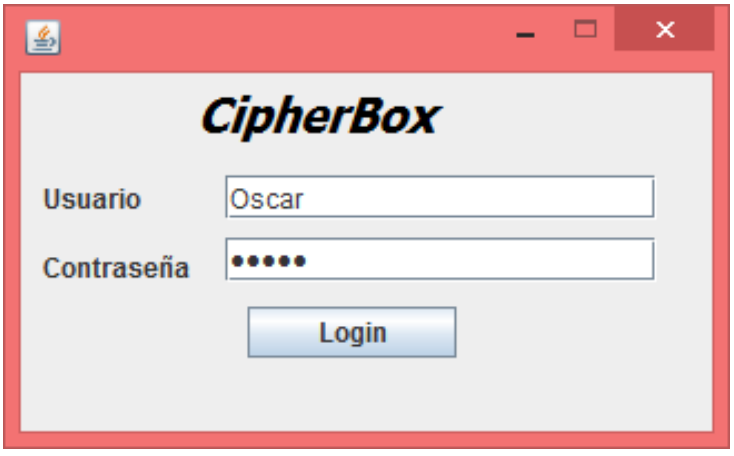

Figura 4.1: Ventana de login

La siguiente ventana, Figura 4.2, será la principal. Desde ella se podrá crear conferencias y unirse a las conferencias a las que estamos invitados. Las nuevas conferencias aparecerán en tiempo real cuando son creadas sin necesidad de refrescar la ventana.

Para crear una conferencia se pulsara el botón que contiene un símbolo con forma de "+", lo cual llevará a la ventana de creación de conferencias, Figura 4.3. En ésta se definirán sus

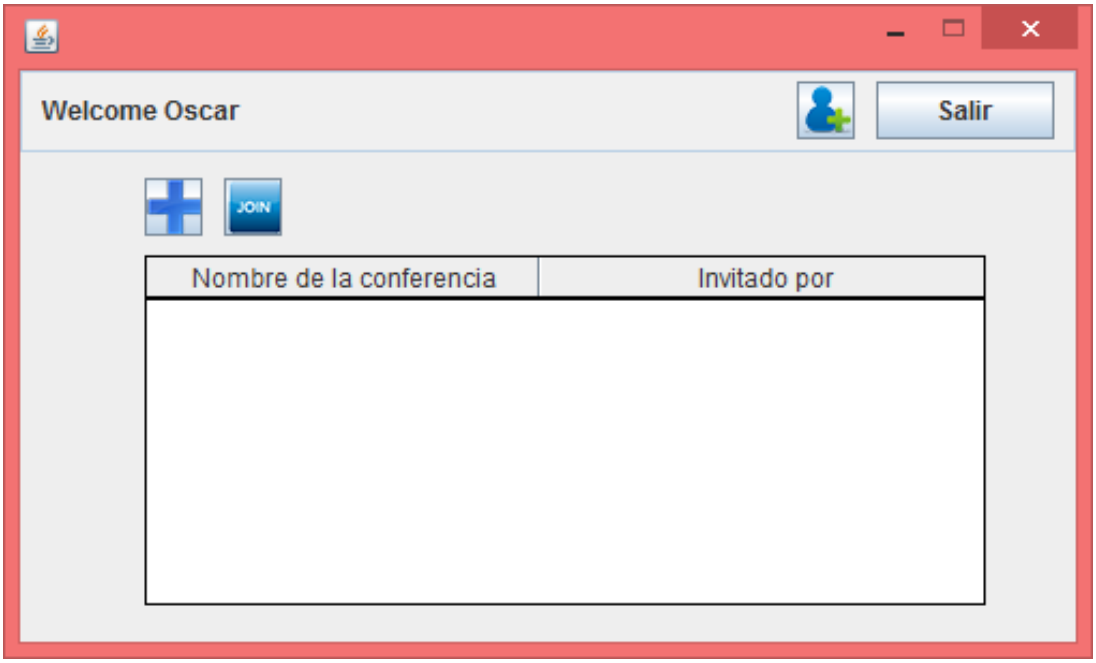

Figura 4.2: Ventana principal

características: el nombre, el protocolo — secuencial o en bloque —, los participantes, así como el tamaño de la matriz y del primo.

Como ya se vio en la comparativa de los protocolos se debe buscar un compromiso entre el tamaño del primo y la matriz. Si se quiere primar la velocidad se eligieran de un tamaño pequeño y si se quiere anteponer la seguridad se usaran valores más elevados.

En esta primera conferencia, creada por Óscar, se usará el protocolo en bloque, un primo de 10 bits y una matriz de cinco por cinco con dos invitados —Alice y Bob—.

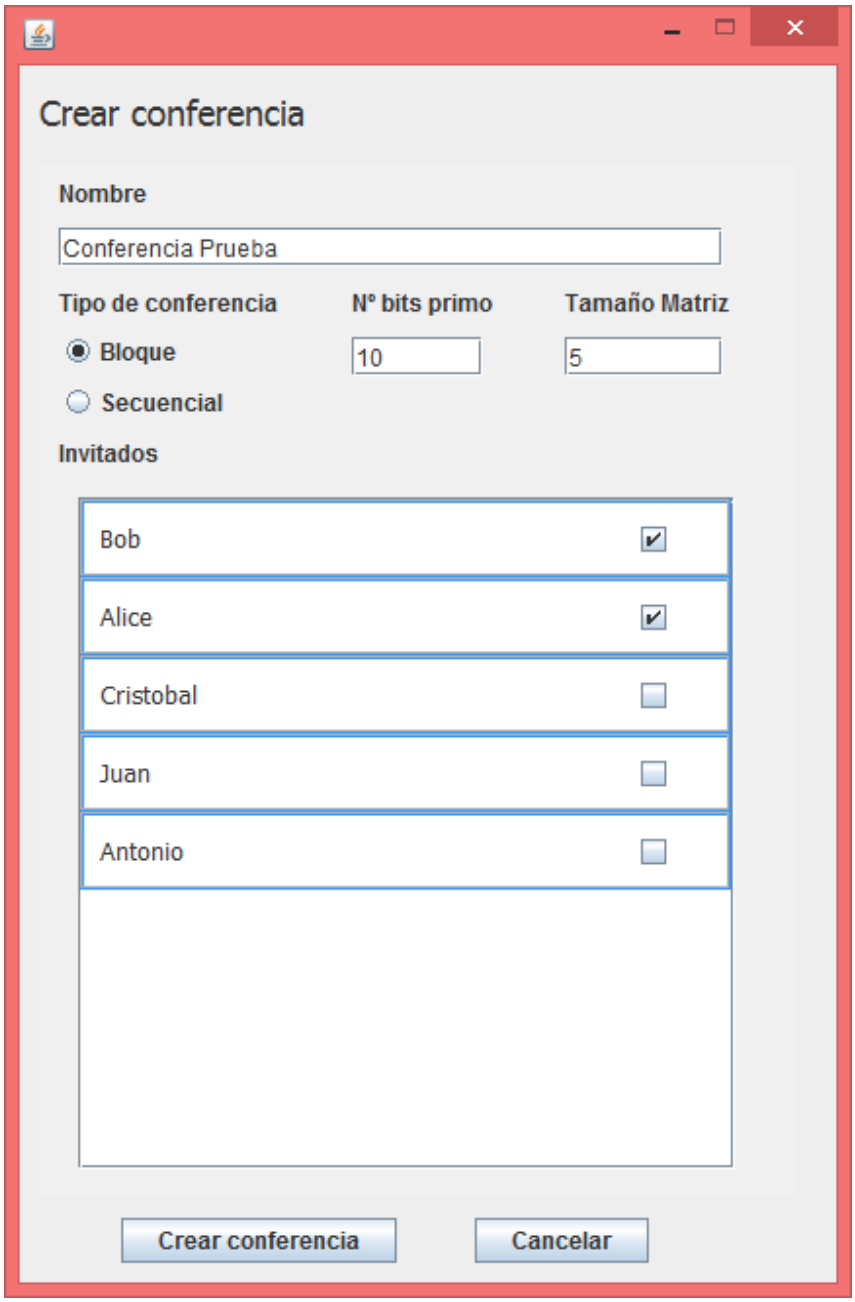

Figura 4.3: Ventana de crear una conferencia.

Después de crear la conferencia se pasará a la siguiente ventana, Figura 4.4. Donde se esperará hasta que el resto de usuarios se una a la conferencia y la clave sea establecida.

Figura 4.4: Ventana de creando conferencia

Ahora cambiando de perceptiva. A Alice, y también a Bob, le aparecerá inmediatamente la invitación a la conferencia en su ventana principal, como se ve en la figura 4.5

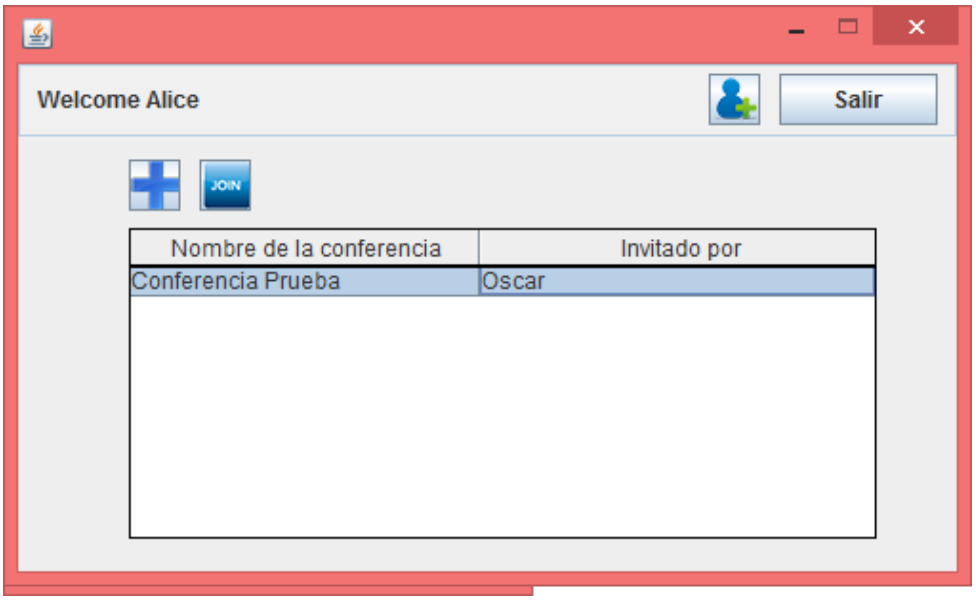

Figura 4.5: Ventana principal de Alice

Para unirse a la conferencia, Alice seleccionará a la conferencia la conferencia entre la lista y pulsará el botón de Join. Entonces pasara a la ventana de uniéndose a la conferencia, Figura 4.6. Donde se esperará al resto de usuarios y se decidirá la calve.

Téngase en cuenta que en este punto se ira avanzado según lo permitan los usuarios que ya se han unido a la conferencia. Por ejemplo si se encuentran los usuarios Óscar, posición 1, y Alice, posición 2, Óscar mandara su vector de elementos K a Alice con los cuales ésta calculara el vector de elementos K que debe mandar a Bob, aunque Bob no se haya unido todavía. Entonces ambos

esperaran hasta que se conecte Bob para poder proseguir con el protocolo.

Por otro lado, si fueran Óscar, posición 1, y Bob ,posición 3, estos no podrían hacer nada hasta que se uniera Alice, posición 2.

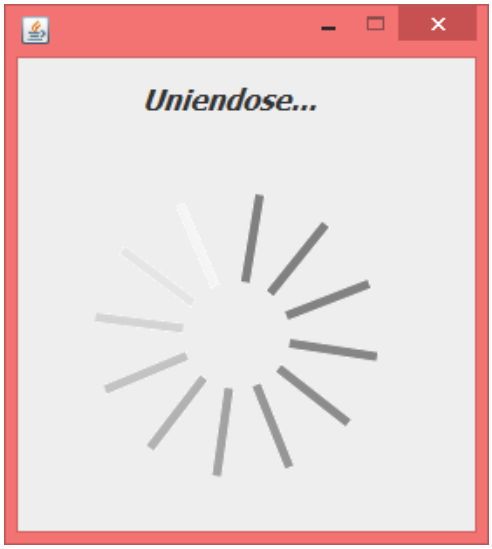

Figura 4.6: Ventana de uniendose a conferencia

Si se entra en la carpeta de la conferencia se puede ver los archivos usados para decidir la clave, Figura 4.7.

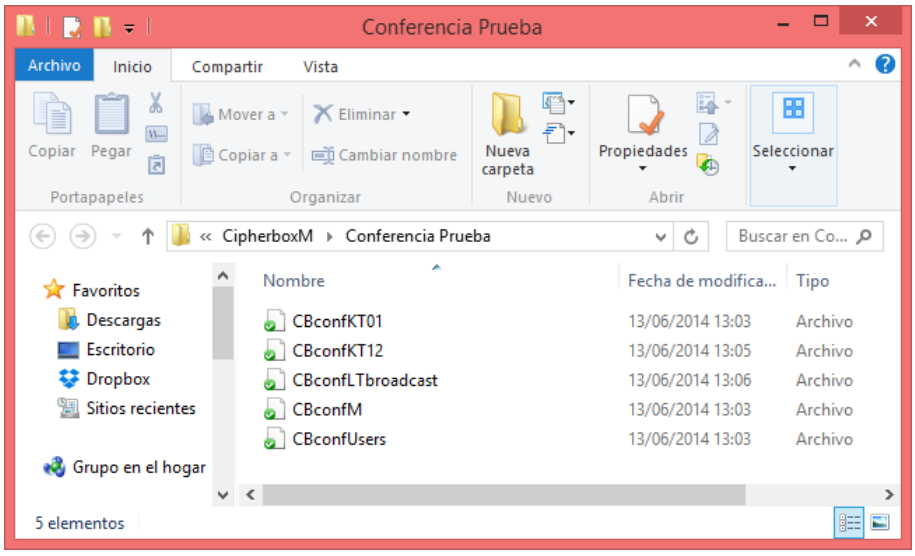

Figura 4.7: Archivos usados durante el intercambio de mensajes entre los usuarios. Bloque

El numero de usuarios es 3 y el protocolo en bloque. Por tanto habrá cuatro ficheros. Un archivo para compartir entre todos los usuarios la matriz M, dos archivos para el intercambio de los vectores de elementos K entre los usuarios — uno para que usuario en la posición 1 lo envié al usuario en la posición 2 y el otro para que el usuario en la posición 2 lo mande al usuario en la posición 3 — y por ultimo, y al ser el protocolo en bloque, un único archivo para mandar el vector de elementos L desde el último usuario al resto.

Una vez todos los usuarios se han unido y, por tanto, se ha decidido la clave secreta de la conferencia se pasa a la ventana de compartición de ficheros, Figura 4.8. Desde ésta cualquier usuario podrá subir, descargar o borrar un fichero y crear directorios. Por último, y como se verá en los posteriores casos de uso, se puede abandonar la conferencia, unir un nuevo usuario a la conferencia o expulsar a un usuario de la conferencia.

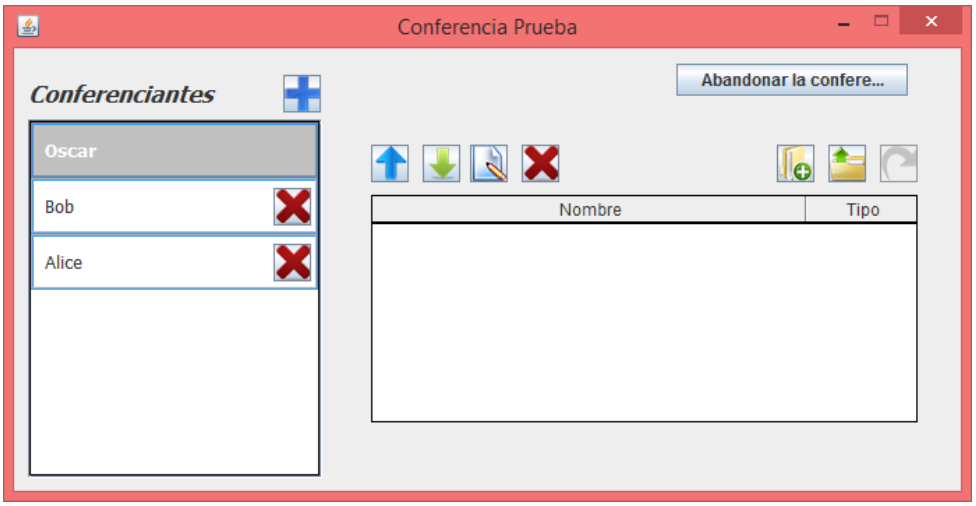

Figura 4.8: Ventana de compartión de ficheros

Ahora se va a probar a compartir un archivo, para ello Bob va a subir el siguiente fichero, Figura 4.9.

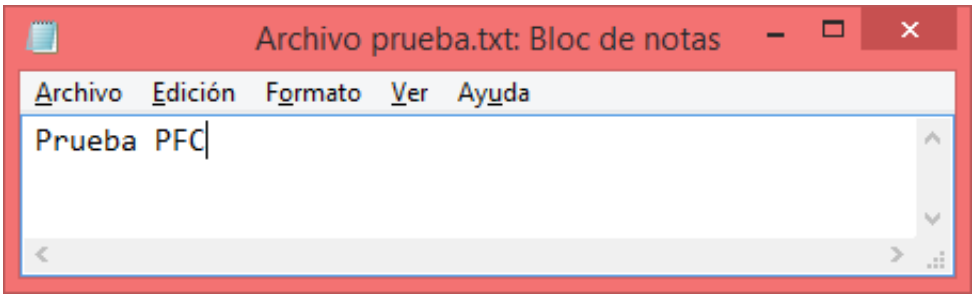

Figura 4.9: Archivo a subir por Bob

Cuando Bob ha subido el fichero, éste aparece inmediatamente al resto de los usuarios, Figura 4.10.

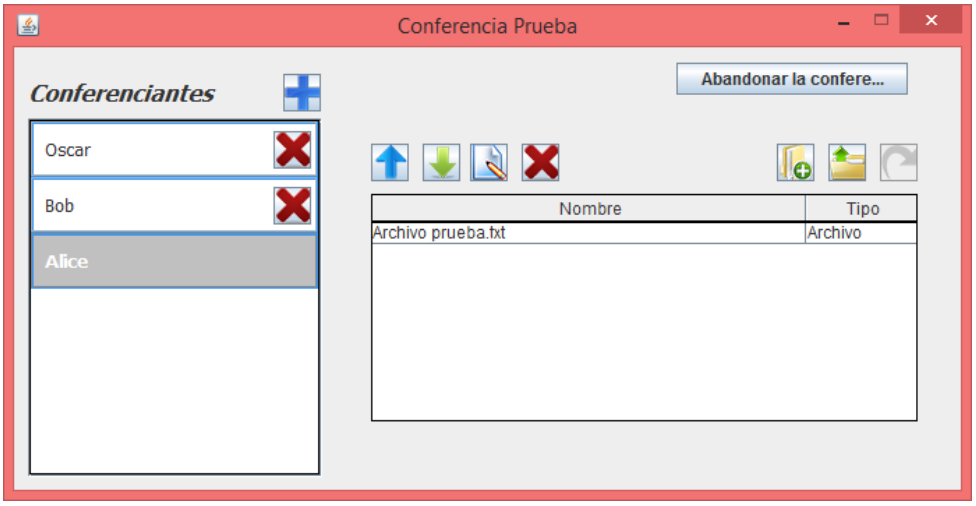

Figura 4.10: Ventana de compartión de ficheros de Alice

Lo siguiente será comprobar el estado que se encuentra el archivo subido por Bob en DropBox.

En la Figura 4.11 se puede ver que ha aparecido un nuevo fichero en DropBox *"Archivo prueba.txt"*.

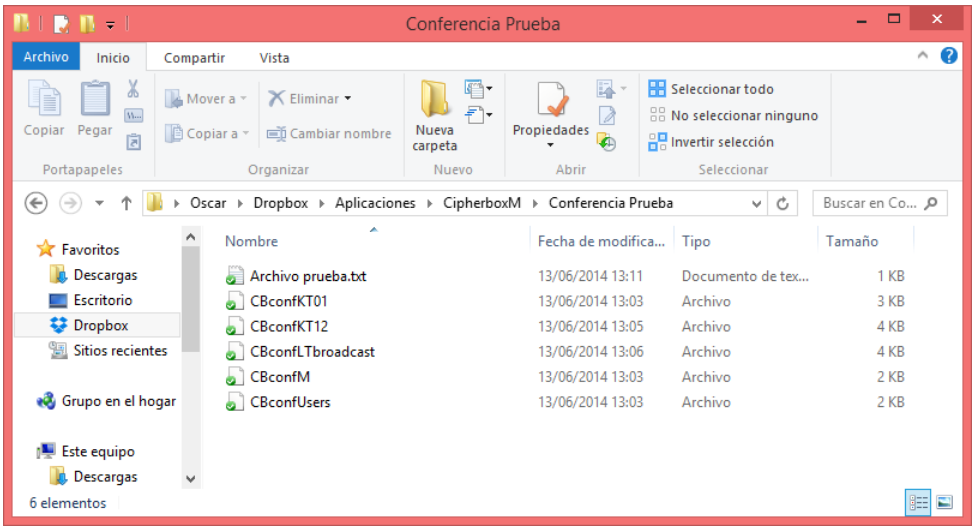

Figura 4.11: Estado de los archivos de DropBox después de subir un archivo

Al abrir este archivo se puede ver que efectivamente está cifrado en DropBox, Figura 4.12.

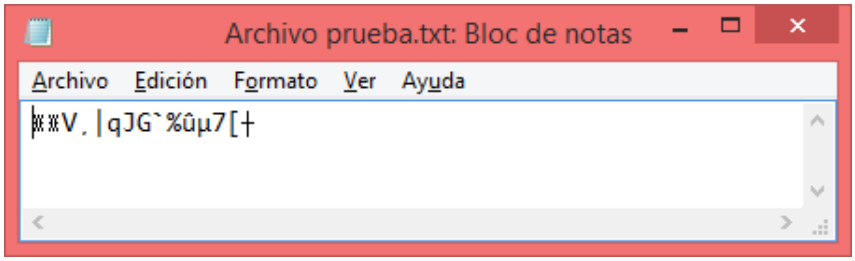

Figura 4.12: Archivo en DropBox

Por ultimo queda que cuando un usuario descarga este archivo de DropBox, se descifrará y será tal y como era el Original. Figura 4.13.

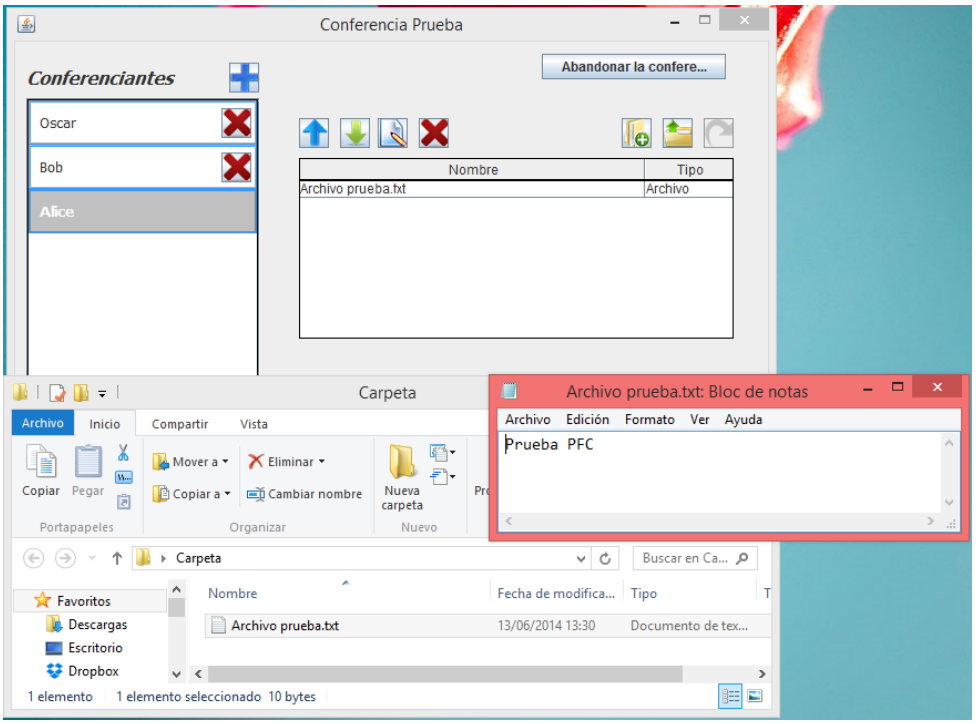

Figura 4.13: Archivo descargado por Alice

4.2. Caso de uso 2: Unir a un usuario

El segundo caso de uso probará la funcionalidad de añadir un nuevo conferenciante a una conferencia ya existente.

Pero antes de nada será necesario crear una conferencia. Para diferir del caso de uso anterior se creara una conferencia usando el protocolo secuencial. El usuario que creará esta conferencia será Alice y los invitados serán Bob y Óscar. El tamaño de la matriz usado será de siete por siete y un primo de 20 bits.

El numero de usuarios es 3 y el protocolo secuencial. Por tanto, se usaran cinco archivos, como se puede ver en la Figura 4.14. Uno para la matriz M, dos para el intercambio del vector de matrices K — uno para que usuario en la posición 1 lo envié al usuario en la posición 2 y el

otro para que el usuario en la posición 2 lo mande al usuario en la posición 3 — y otros dos para el intercambio de matrices L — uno para que usuario en la posición 3 lo envié al usuario en la posición 2 y el otro para que el usuario en la posición 2 lo mande al usuario en la posición 1 —.

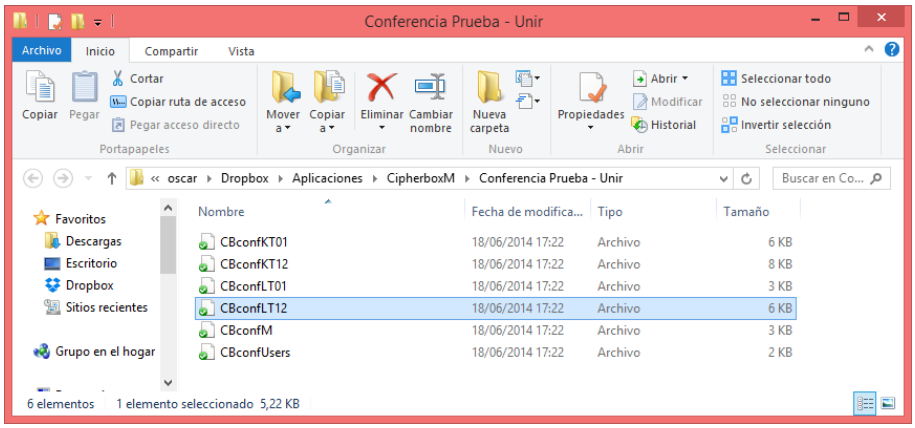

Figura 4.14: Archivos usados durante el intercambio de mensajes entre los usuarios. Secuencial

Una vez creada la conferencia se va a pasar a probar la funcionalidad de añadir un nuevo usuario a la conferencia. Para ello uno de los usuarios, en este caso Óscar, deberá pulsar el botón con el símbolo de "+". Entonces pasara a la ventana de unir usuarios. Notese que solo se muestran los usuarios que actualmente no forman parte de la conferencia. Finalmente se elegirá al usuario que se quiere añadir(Juan) a la conferencia.

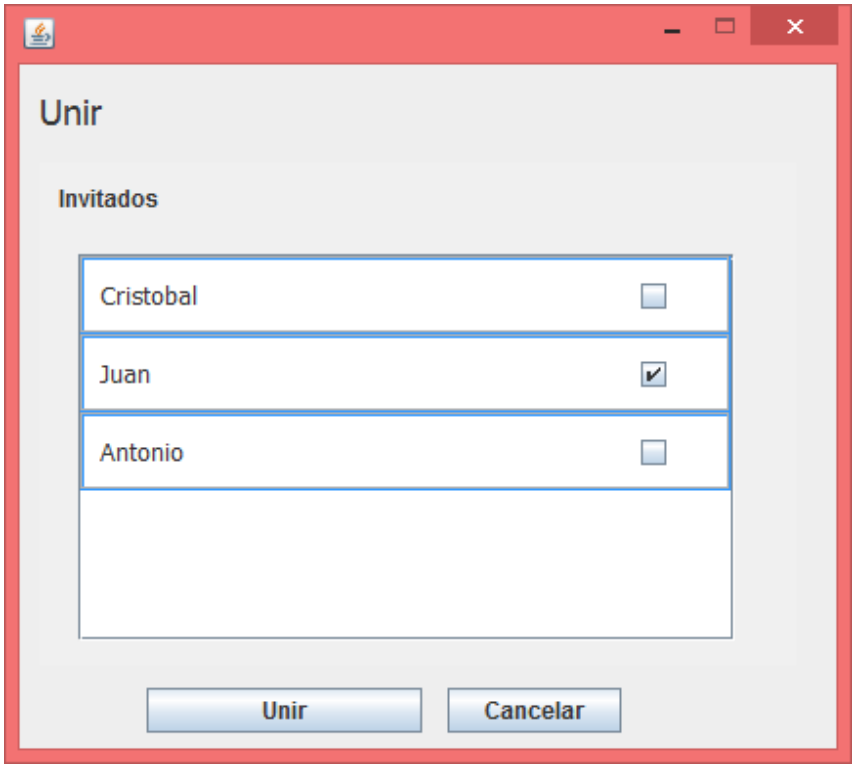

Figura 4.15: Ventana de unir usuario a la conferencia

Todos los usuarios de la conferencia pasaran a la ventana de regeneración de la clave, Figura 4.16, donde esperaran hasta que el nuevo usuario, Juan, se una a la conferencia.

Figura 4.16: Ventana de regeneración a conferencia

La nueva conferencia le aparecerá a Juan en su ventana principal, Figura 4.17, desde donde Juan podrá unirse a la conferencia.

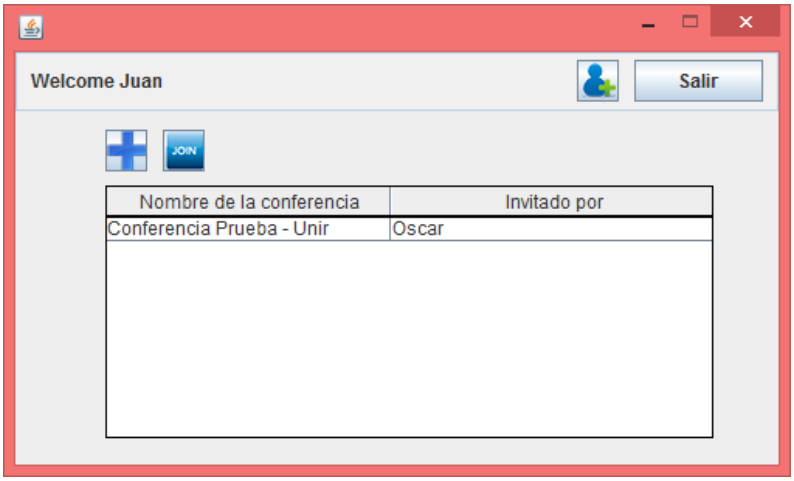

Figura 4.17: Ventana principal de Juan

Una vez en la ventana de uniéndose a la conferencia, Juan decidirá la nueva clave con el resto de usuarios(que se encuentran en la ventana de regenerando la clave) y una vez decidida pasaran de nuevo a la ventana de compartición de ficheros, Figura 4.18.

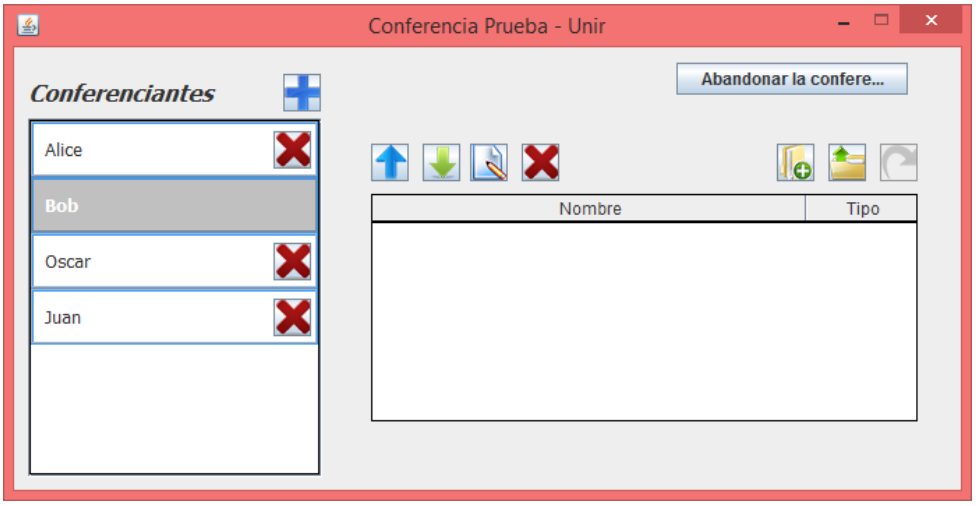

Figura 4.18: Ventana de compartión de ficheros despues de unir al nuevo usuario

Al haber subido el numero de usuarios de la conferencia de 3 a 4, por tanto, (y debido al estar usando el protocolo secuencial) se usaran 7 ficheros en lugar de 5, Figura 4.19. Los dos nuevos archivos añadidos serán para enviar el vector de elementos K desde el que antes era el ultimo usuario al nuevo y enviar el vector de elementos L desde el nuevo usuario al que antes era el último. Si se estuviera usando el protocolo en bloque solo hubiera sido necesario añadir un único fichero, puesto que el vector de elementos L se mandará mediante un archivo ya existente desde el nuevo usuario al resto.

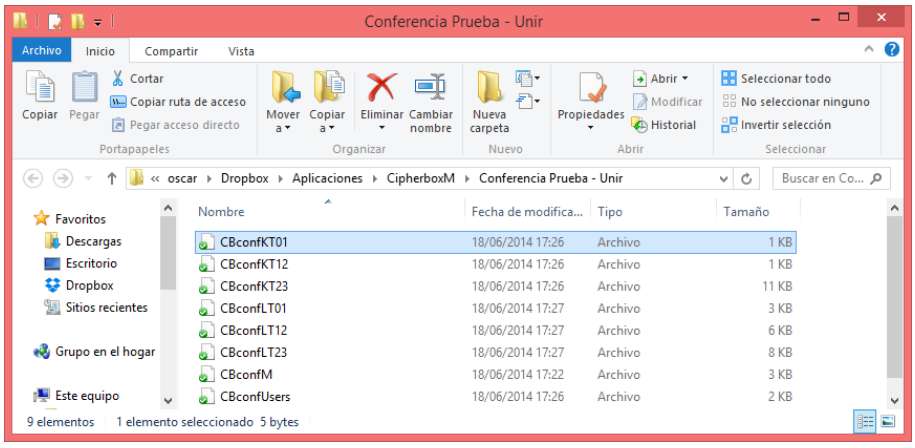

Figura 4.19: Archivos usados durante el intercambio de mensajes entre los usuarios. Secuencial

Para comprobar que los usuarios siguen compartiendo la misma clave se va a compartir un fichero. A diferencia del caso de uso anterior donde se utilizo un archivo de texto plano, en éste se va a utilizar una imagen (Figura 4.20) para probar que se pueden compartir ficheros de toda índole.

Figura 4.20: Imagen a compartir

Se puede comprobar que la imagen aparece inteligible en DropBox, Figura 4.21.

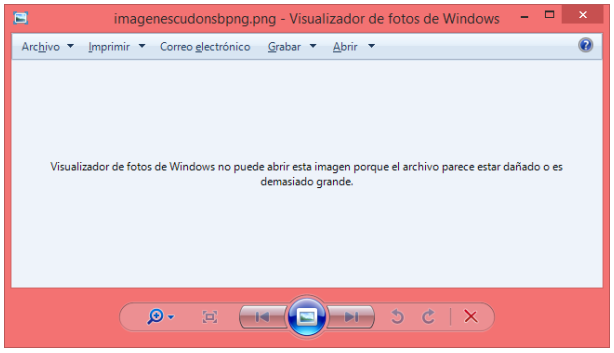

Figura 4.21: Imagen en DropBox

Por último queda descargar la imagen desde otro usuario, Juan, y comprobar que es igual a la original. Figura 4.22.

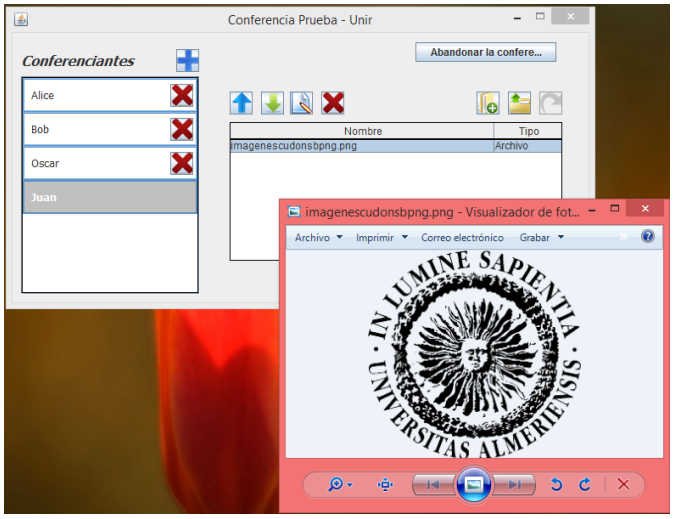

Figura 4.22: Imagen descargado por Juan

4.3. Caso de uso 3: Expulsar un usuario

Para este caso de uso se partirá del estado dejado al final del anterior. En éste se probará la funcionalidad de expulsar a un usuario de una conferencia.

Para ello bastará con pulsar el botón con el símbolo de "xrojo situado a la derecha del nombre de cada usuario. Al pulsar este botón, se informará al usuario expulsado que ha sido expulsado, 4.23, y el resto de usuarios pasará a la ventana de regeneración de la clave.

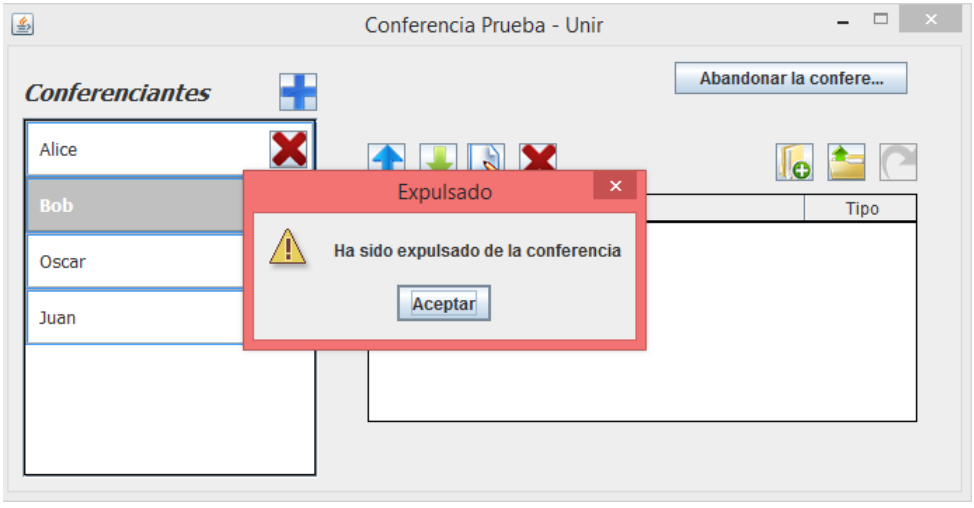

Figura 4.23: Aviso de usuario expulsado

En este caso no habrá que esperar a ningún otro usuario, por tanto los usuarios empezaran inmediatamente el procedimiento para regenerar la clave y volverán a la ventana de compartición de ficheros, Figura 4.24.

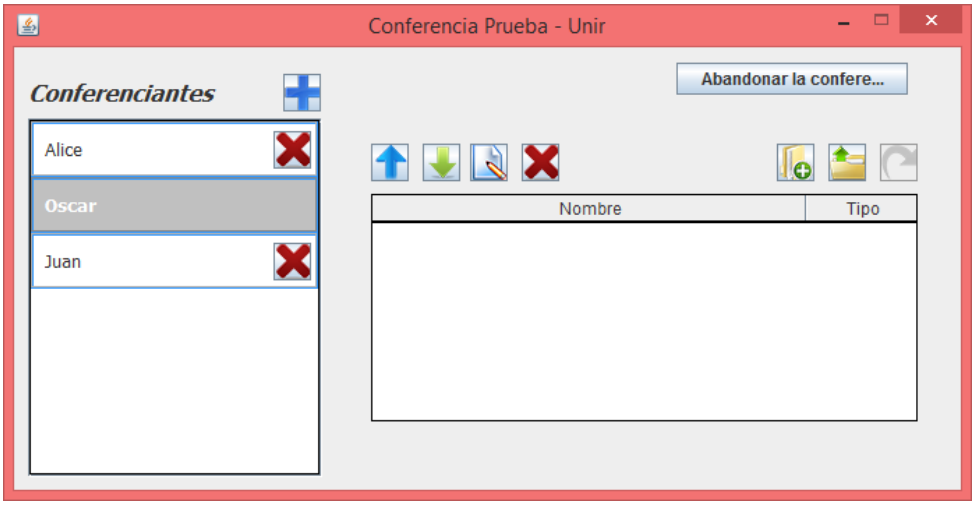

Figura 4.24: Ventana de compartión de ficheros despues de expulsar a Bob

Al haber bajado de cuatro a tres usuarios, también se verán reducidos los archivos necesarios para realizar la comunicación como se ve en la Figura 4.25. Los archivos vuelven a ser 5: la matriz M; dos para los vectores K entre los usuarios; dos para los vectores L entre los usuarios.

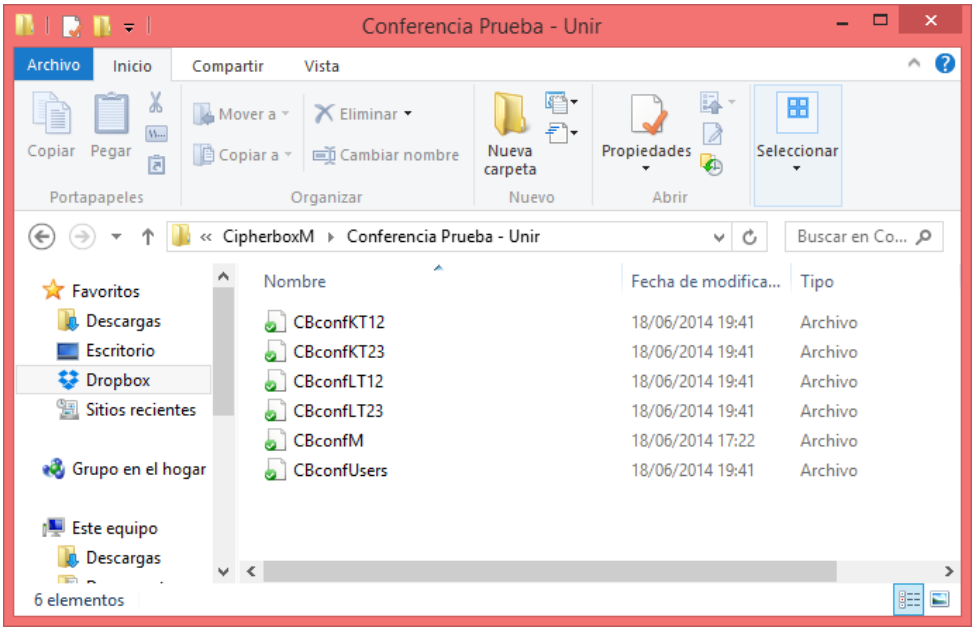

Figura 4.25: Archivos despues de expulsar a Bob

De nuevo se va a comprobar que los usuarios siguen compartiendo la misma clave. Para ello se va a aprovechar y se probará una funcionalidad todavía no usada, la de crear, eliminar y navegar por directorios.

Lo primero será crear un directorio, Figura 4.26.

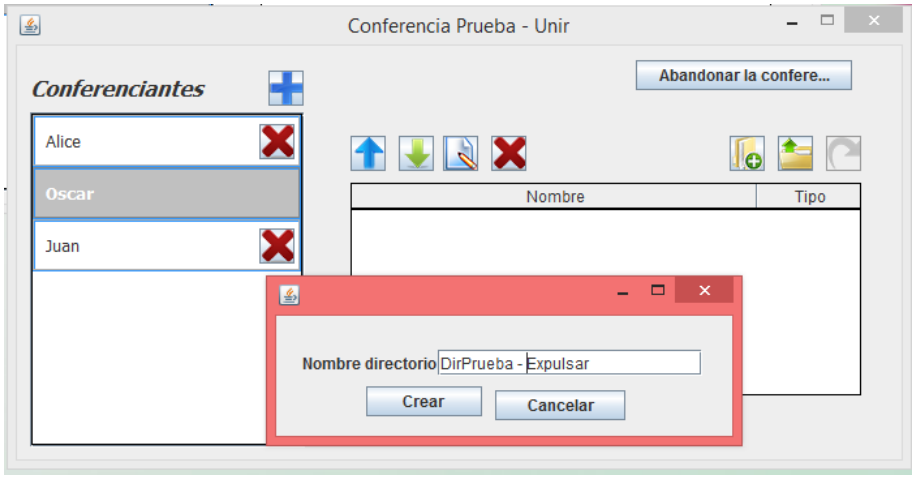

Figura 4.26: Crear directorio

Y como sucedía con los archivos, éste aparecerá automáticamente a todos los usuarios, Figura 4.27, bajo el tipo directorio.

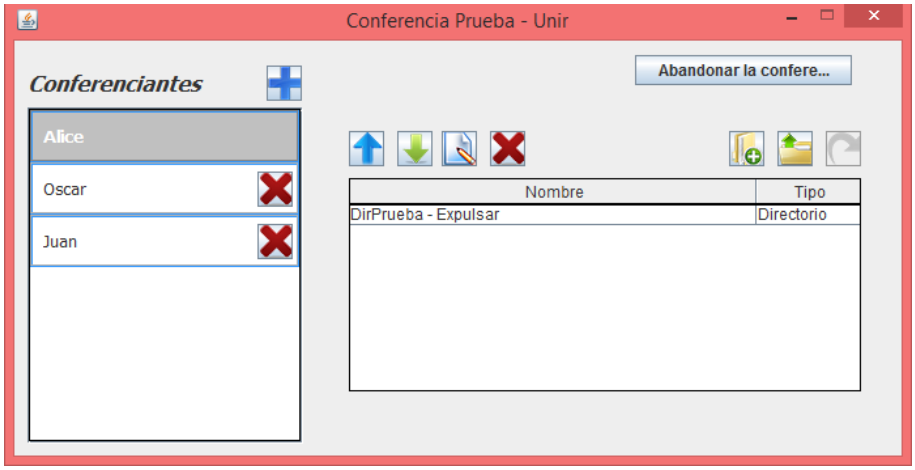

Figura 4.27: Ventana de compartición de ficheros con directorio

El directorio sera representado como una carpeta dentro de la carpeta de la conferencia, Figura 4.28. En ésta se podrán realizar las mismas operaciones que se podían realizar sobre la carpeta de la conferencia.

Para comprobar que el sistema de directorios funciona y que los usuarios aun comparten la misma clave se va a proceder a subir la imagen que se utilizó en el anterior caso de uso. Y como se ve en la, Figura 4.28, ésta aparece en DropBox cifrada dentro del directorio creado.

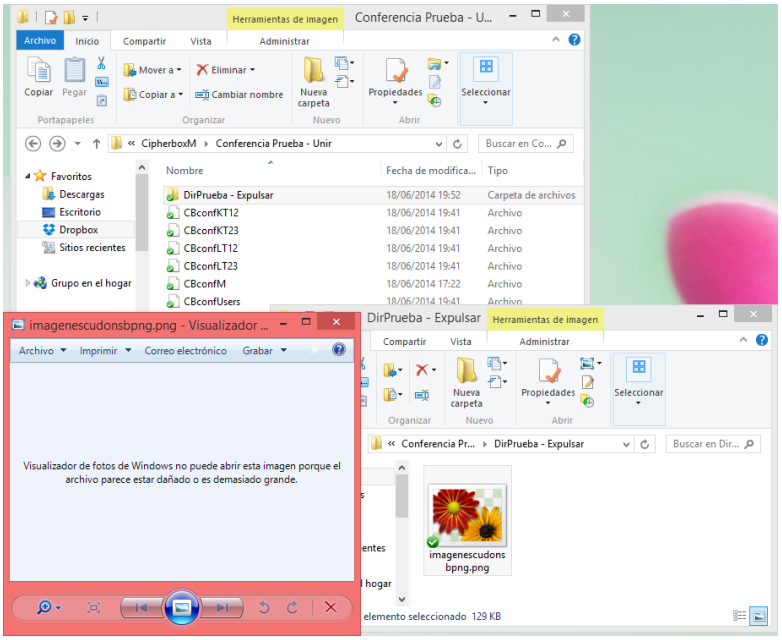

Figura 4.28: Archivos en DropBox

Por último se comprueba que al descargar la imagen desde otro usuario ésta es la misma que se subió. Figura 4.29

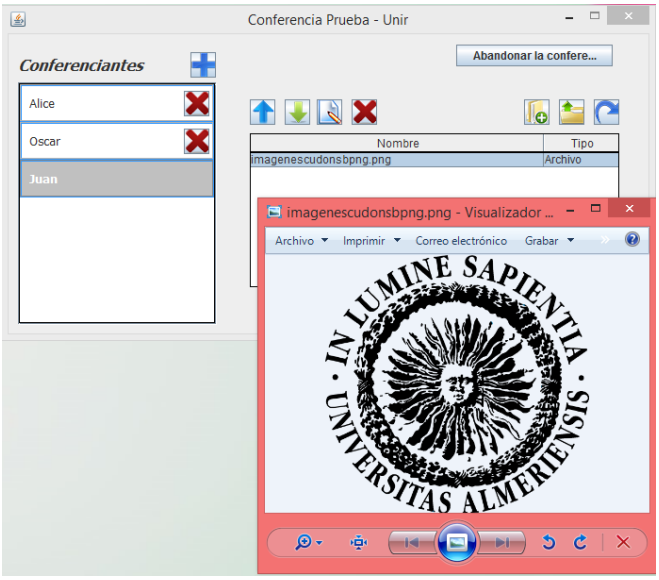

Figura 4.29: Imagen descargada por Juan

4.4. Caso de uso 4: Abandonar la conferencia

Para este caso de uso se partirá del estado dejado al final del anterior. El último caso de uso tratará sobre la funcionalidad de abandonar la conferencia, que puede ser vista expulsarse a uno mismo. Por lo que el protocolo y metodología seguida será las mismas que las usadas para la expulsión.

En este caso el usuario que abandonara la conferencia será Alice, quien la creó. Esto se hace con dos propósitos: comprobar el funcionamiento de la funcionalidad de abandonar la conferencia y probar que una conferencia puede proseguir después de que su creador la abandone.

Para ello Alice pulsará el botón de *"Abandonar la conferencia"* o la *"x"* de la ventana. Entonces el resto de usuarios pasarán a la ventana de regenerar la clave, mientras que Alice volverá a la ventana principal. Cuando los usuarios restantes de la conferencia hayan decidido una nueva clave volverán a la ventana de compartición de ficheros, Figura 4.30.

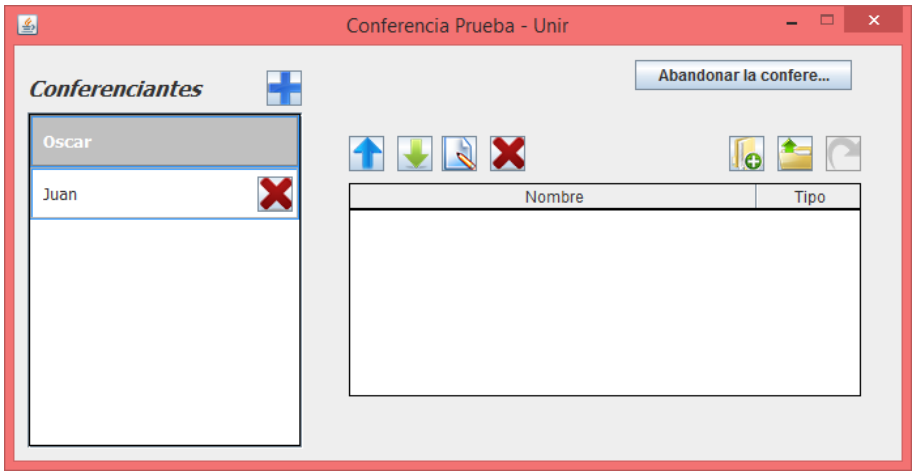

Figura 4.30: Ventana de compartición de ficheros despues del que Alice abandone la conferencia

Como sucedió en el caso anterior, al bajar de tres usuarios a dos, también bajará el numero de ficheros necesarios para realizar la comunicación como se refleja en la figura 4.31. Siendo solo necesarios tres archivos: una para la matriz M, otro para el intercambio del vector de elementos K y uno último para el vector de elementos L.

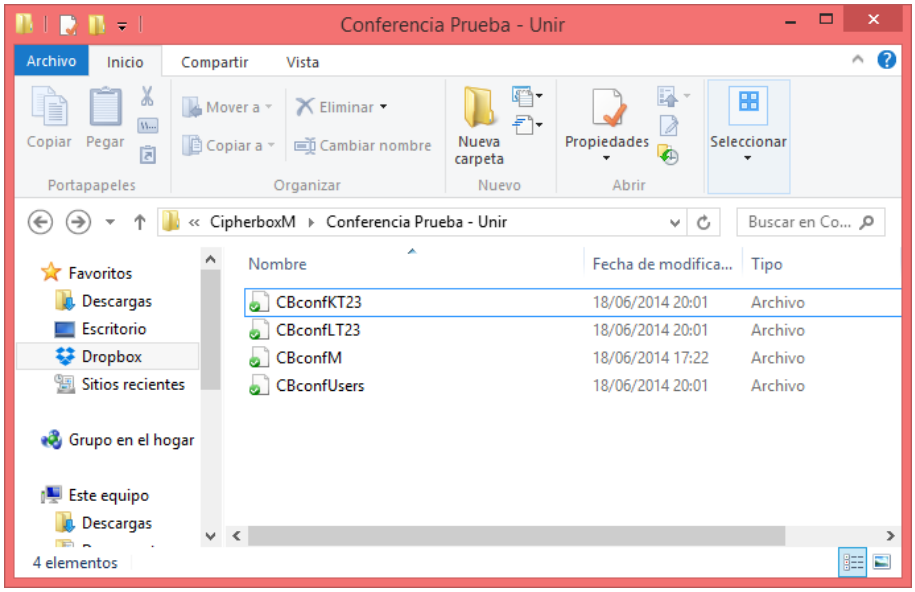

Figura 4.31: Archivos en DropBox

Finalmente se procederá a terminar la conferencia. Para ello como solo quedan dos usuarios bastara con que uno expulse al otro o uno de ellos abandone la conferencia. En este caso será Óscar quien abandone la conferencia volviendo a la ventana principal. Mientras a Bob se le notificará que la conferencia ha concluido pues no quedan mas usuarios, Figura 4.32.

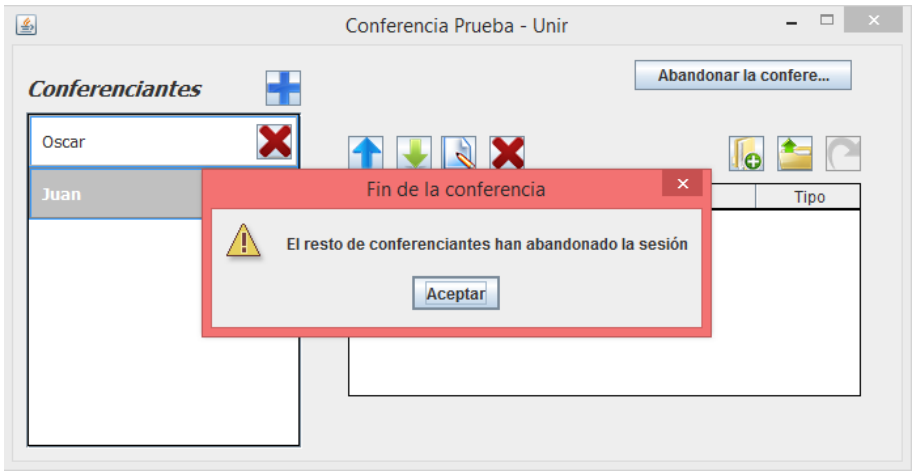

Figura 4.32: Aviso del final de la conferencia

Al terminar una conferencia se borraran los ficheros subidos, directorios creados, el archivo de configuración, los archivos para intercambios de claves, etc. En definitiva se eliminará todo rastro de la existencia de la conferencia de DropBox como se puede comprobar en la Figura 4.33.

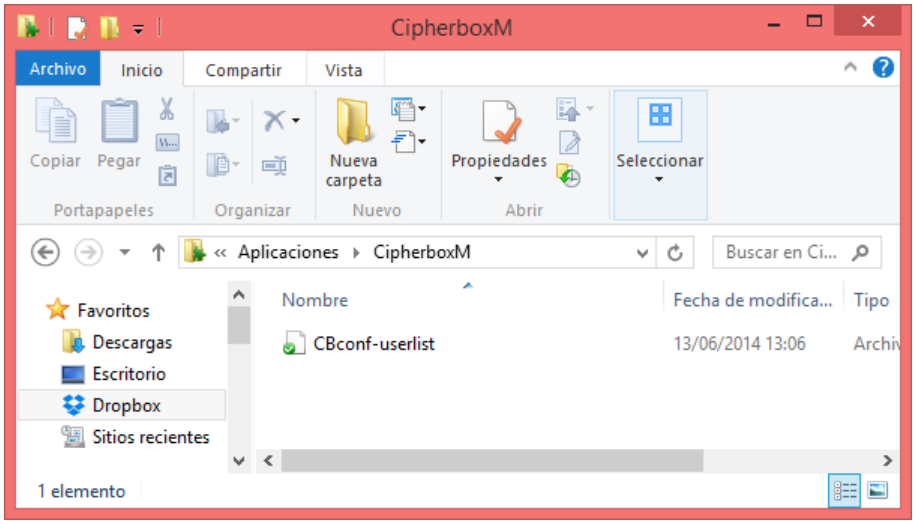

Figura 4.33: Los rastros de la conferencia han sido borrados de DropBox

Conclusiones y trabajos futuros

Conclusiones

En este proyecto se han implementado dos protocolos de multidifusión de claves basados en un problema algebraico no conmutativo (el programa de la descomposición) demostrados teóricamente. Con el objetivo de probar que efectivamente es posible realizar una implementación de estos y, finalmente, para desarrollar una plataforma de intercambio seguro de ficheros entre grupos usuarios.

El protocolo secuencial será más aconsejable para conferencias con un número reducido de usuarios debido a que requiere de una menor cantidad de cálculos a la hora de decidir la clave secreta que el protocolo en bloque. Por otro lado el protocolo en bloque resulta mas aconsejable para conferencias con un elevado número de usuarios . Puesto que el secuencial necesita de aproximadamente el doble de mensajes a través de la red para decidir la clave que el protocolo en bloque. El protocolo secuencial usará 2n-1 mensajes, siendo n el número de usuarios, mientras el protocolo en bloque únicamente n+1 mensajes.

En ambos protocolos será necesario buscar un compromiso entre el tamaño en bits del número primo y el tamaño de la matriz. Cuantos mas elevados sean estos valores mas difícil será el obtener la clave secreta mediante un ataque de fuerza bruta. Pero si se utilizan valores muy elevados para ambos, la cantidad y el tamaño de los datos que se manejarán aumentarán rápidamente y en consecuencia también lo hará el tiempo.

Un ejemplo de esto es que si se utilizara un número primo no muy elevado, 503, y una matriz de un tamaño de 10x10. Se tendrán 100 valores, donde en la última fila los números podrán alcanzar valores de hasta $503^{10} = 1036763861454476003909724049$.

Por otro lado se ha probado que toda la comunicación necesaria para realizar la multidifusión puede realizarse usando la API de DropBox. Pero pese a ser posible esta API estaba pensada para el intercambio únicamente de ficheros y no cuenta con herramientas para el intercambio de estructuras de datos y para su sincronización entre los usuarios. Por lo que ha sido necesario solventar estas carencias utilizando junto a la API de DropBox otras herramientas que proporciona Java y utilizar estructuras de datos más complejas para hacer esta implementación posible.

Trabajos futuros

El primero de los trabajos futuros tiene que ver con la API DataStore[3] de DropBox. Esta API está enfocada a la sincronización más que únicamente los archivos. Ademas permite sincronizar estructuras de datos, resolución automática de conflictos, almacenes de datos y tablas en las que realizar consultar.

La API DataStore solventa la mayor parte de los problemas encontrados con el uso API de DropBox de una forma sencilla y con poco esfuerzo. Ademas al ser una herramienta única dedicada a la sincronización obtendrá mejores resultados que los que se han obtenido en esta implementación. Debido a que no se necesitarán usar herramientas adicionales sobre la API ni estructuras de datos complejas para la gestión, intercambio e sincronización, ya que la API proporcionará todo esto.

Otra linea de trabajo futuro consiste en buscar conseguir un *speed up* en el tiempo de las operaciones matemáticas utilizadas para decidir la clave. La primera linea de mejora será el mejorar la operación matemática mas utilizada, la multiplicación de matrices. Debido a que al estar presente en todas las etapas del protocolo una pequeña mejora en ésta supondrá una mejora considerable.

El primer enfoque para buscar esta mejora será en optar por una implementación paralela mediante hebras y usando las técnicas de programación dinámica frente a la versión secuencial actual.

Otra operación cuya mejora podría suponer un *speed up* es la del exponencial de una matriz a un número entero. $f(x)^r mf(x)^s$. Ésta también se podría mejorar mediante su paralización.

También podrían realizarse en realizar una implementación en dispositivos móviles. Puesto que se adaptan fácilmente a la filosofía del protocolo y la migración a Android de la base matemática implementada será sencilla .

Por último, el protocolo implementado de multidifusión de claves podrá ser utilizada en todo tipo de aplicaciones y no solo para el intercambio de ficheros. Sistemas de mensajería, videollamadas y en definitiva en cualquier tipo de aplicación donde un grupo de usuarios necesite comunicarse.

Apéndices

Apéndice A

Ejemplos de test unitarios

Nota sobre los apéndices: en los apéndices he puesto aquello de la implementación que no esta del todo relacionado directamente con el protocolo o resulta de menos importancia. No sé si sera adecuado poner estos puntos en la documentación, pero los dejo y si consideráis que alguno no es adecuado me resulta mucho mas rápido quitarlo que añadirlo.

Código A.1: Test del formato de la matriz

```
1   @Test
2   public void checkMatrixFormatTest() {
3       Matriz m1 = new Matriz(
4               "src/com/ual/pfc/content/matriz/examplesMatriz/m1.txt");
5
6       assertTrue(m1.checkMatrizFormat());
7
8       Matriz m2 = new Matriz(
9               "src/com/ual/pfc/content/matriz/examplesMatriz/m2.txt");
10
11      assertTrue(m2.checkMatrizFormat());
12
13      Matriz mrandom = new Matriz(
14              "src/com/ual/pfc/content/matriz/examplesMatriz/mrandom.txt");
15
16      assertTrue(mrandom.checkMatrizFormat());
17
18      Matriz maux = new Matriz(
19              "src/com/ual/pfc/content/matriz/examplesMatriz/maux.txt");
20
21      assertTrue(maux.checkMatrizFormat());
22
23      Matriz mIF = new Matriz(
24              "src/com/ual/pfc/content/matriz/examplesMatriz/mIncorectFormat.txt");
25
26      assertTrue(!mIF.checkMatrizFormat());
27
28  }
```

Código A.2: Test de generación de matrices aleatorias

```
1
2       @Test
3       public void randomPublicMatrizM1Test() {
4           Matriz m;
5
6           for (int i = 0; i < 15; i++) {
7               m = new MatrizClavePublica(5, 5, new BigInteger("3"));
8
9               assertTrue(m.checkMatrizFormat());
10          }
11
12          for (int i = 0; i < 15; i++) {
13              m = new MatrizClavePublica(7, 7, new BigInteger("5"));
14              assertTrue(m.checkMatrizFormat());
15          }
16
17      }
```

Código A.3: Test de multiplicación de dos matrices

```
1       @Test
2       public void m1multiplym2Test() {
3           Matriz m1 = new Matriz(
4                   "src/com/ual/pfc/content/matriz/examplesMatriz/m1.txt");
5           Matriz m2 = new Matriz(
6                   "src/com/ual/pfc/content/matriz/examplesMatriz/m2.txt");
7
8           Matriz mResult = m1.multiply(m2);
9           Matriz mExpetedResult = new Matriz(
10                  "src/com/ual/pfc/content/matriz/examplesMatriz/m1bym2.txt");
11
12          assertTrue(mResult.isEqual(mExpetedResult));
13      }
```

Código A.4: Test del funcionamiento protocolo sin encapsular los usuarios

```
1       @Test
2       public void intercambioTest() {
3
4           Extras e = new Extras();
5
6           MatrizClavePublica m = new MatrizClavePublica(
7                   "src/com/ual/pfc/content/examplePolinomi/m.txt");
8           MatrizClavePublica n = new MatrizClavePublica(
9                   "src/com/ual/pfc/content/examplePolinomi/n.txt");
10
11          Polinomio f = (Polinomio) e
12                  .getSerializar("src/com/ual/pfc/content/examplePolinomi/f");
13          Polinomio g = (Polinomio) e
14                  .getSerializar("src/com/ual/pfc/content/examplePolinomi/g");
15
16          int r = 5;
```

```
17          int s = 7;
18
19          int u = 9;
20          int v = 8;
21
22          Matriz fM = f.aplicarFuncion(m);
23          Matriz gM = g.aplicarFuncion(m);
24
25          Matriz clavePublicaAlice = fM.pow(r).multiply(n).multiply(fM.pow(s));
26          clavePublicaAlice
27                  .toFile("src/com/ual/pfc/content/examplePolinomi/alicePublica");
28          Matriz clavePublicaBob = gM.pow(u).multiply(n).multiply(gM.pow(v));
29          clavePublicaBob
30                  .toFile("src/com/ual/pfc/content/examplePolinomi/bobPublica");
31
32          Matriz sA = fM.pow(r).multiply(clavePublicaBob).multiply(fM.pow(s));
33          Matriz sB = gM.pow(u).multiply(clavePublicaAlice).multiply(gM.pow(v));
34
35          assertTrue(sA.isEqual(sB));
36          sA.toFile("src/com/ual/pfc/content/examplePolinomi/secreto");
37
38      }
```

Código A.5: Test del protocolo secuencial

```
1       @Test
2       public void intercambioNUserTest() {
3
4           ArrayList<UserBacktrack> usuarios = new ArrayList<UserBacktrack>();
5
6           int N = 10;
7
8           for (int i = 0; i < N; i++) {
9               if (i == 0) {
10                  usuarios.add(new UserBacktrack());
11              } else if (i == N - 1) {
12                  usuarios.add(new UserBacktrack(usuarios.get(i - 1).getM(),
13                          usuarios.get(i - 1).getClavesPublicas(), true));
14
15              } else {
16                  usuarios.add(new UserBacktrack(usuarios.get(i - 1).getM(),
17                          usuarios.get(i - 1).getClavesPublicas(), false));
18              }
19          }
20
21          for (int i = N - 2; i ≥0; i--) {
22              usuarios.get(i).crearClaveConversacion(usuarios.get(i + 1).getL());
23          }
24
25          for (int i = 1; i < N; i++) {
26              assertTrue(usuarios.get(0).getClaveConversacion()
27                      .isEqual(usuarios.get(i).getClaveConversacion()));
28
29          }
30
```

```
31      }
```

Código A.6: Test del protocolo en bloque

```
1       @Test
2       public void intercambioNUserTest() {
3
4           ArrayList<UserBroadcast> usuarios = new ArrayList<UserBroadcast>();
5
6           int N = 10;
7
8           for (int i = 0; i < N; i++) {
9               if (i == 0) {
10                  usuarios.add(new UserBroadcast());
11              } else if (i == N - 1) {
12                  usuarios.add(new UserBroadcast(usuarios.get(i - 1).getM(),
13                          usuarios.get(i - 1).getClavesPublicas(), true));
14
15              } else {
16                  usuarios.add(new UserBroadcast(usuarios.get(i - 1).getM(),
17                          usuarios.get(i - 1).getClavesPublicas(), false));
18              }
19          }
20
21
22          for (int i = N - 2; i ≥0; i--) {
23              usuarios.get(i).crearClaveConversacion(usuarios.get(N-1).getL());
24          }
25
26          for (int i = 1; i < N; i++) {
27              assertTrue(usuarios.get(0).getClaveConversacion()
28                      .isEqual(usuarios.get(i).getClaveConversacion()));
29
30          }
31      }
```

Código A.7: Test de cifrado y descifrado de documentos

```
1       @Test
2       public void cifradoDescifradoTextoTest() {
3
4           String texto = "Prueba de cifrado y descifrado";
5
6           UserBroadcast Alice = new UserBroadcast();
7           UserBroadcast Bob = new UserBroadcast(Alice.getM(),
8                   Alice.getClavesPublicas(), true);
9
10
11
12          Alice.crearClaveConversacion(Bob.getL());
13
14          assertTrue(Alice.getClaveConversacion().isEqual(
15                  Bob.getClaveConversacion()));
16
```

```
17          byte[] cifrado;
18          cifrado = Alice.cifrar(texto.getBytes());
19          String descifrado = new String(Bob.descifrar(cifrado));
20          assertEquals(texto, descifrado);
21
22      }
```

Código A.8: Test de unir conferenciante en protocolo secuencial

```
1       @Test
2       public void unirseConferenciaTest() {
3           UserBacktrack Alice=new UserBacktrack();
4           UserBacktrack Bob=new UserBacktrack(Alice.getM(),Alice.getClavesPublicas(),true);
5
6           Alice.crearClaveConversacion(Bob.getL());
7
8           assertTrue(Alice.getClaveConversacion().isEqual(Bob.getClaveConversacion()));
9
10          //unir nuevo usuario
11          Bob.actualizarClavePrivada(false);
12
13          UserBacktrack Oscar=new UserBacktrack(Bob.getM(),Bob.getClavesPublicas(),true);
14
15          Bob.crearClaveConversacion(Oscar.getL());
16          Alice.crearClaveConversacion(Bob.getL());
17
18          //coprobamos que todos los usuarios tienen la misma clave
19          assertTrue(Alice.getClaveConversacion().isEqual(
20                  Bob.getClaveConversacion()));
21          assertTrue(Alice.getClaveConversacion().isEqual(
22                  Oscar.getClaveConversacion()));
23      }
```

Código A.9: Test de expulsar conferenciante en protocolo en bloque

```
1       @Test
2       public void expulsarConferenciaTest() {
3
4           UserBroadcast Alice = new UserBroadcast();
5           UserBroadcast Bob = new UserBroadcast(Alice.getM(),
6                   Alice.getClavesPublicas(), false);
7           UserBroadcast Oscar = new UserBroadcast(Bob.getM(),
8                   Bob.getClavesPublicas(), false);
9           UserBroadcast Juan = new UserBroadcast(Oscar.getM(),
10                  Oscar.getClavesPublicas(), true);
11
12          Oscar.crearClaveConversacion(Juan.getL());
13          Bob.crearClaveConversacion(Juan.getL());
14          Alice.crearClaveConversacion(Juan.getL());
15
16          assertTrue(Alice.getClaveConversacion().isEqual(
17                  Bob.getClaveConversacion()));
18          assertTrue(Alice.getClaveConversacion().isEqual(
19                  Oscar.getClaveConversacion()));
```

```
20          assertTrue(Alice.getClaveConversacion().isEqual(
21                  Juan.getClaveConversacion()));
22
23          //vamos a explusar a Bob -> expulsar a un usuario del medio
24
25          Alice.actualizarClavePrivada(false);
26
27          Oscar.actualizarClavesPublicas(Alice.getClavesPublicas(),false);
28          Juan.actualizarClavesPublicas(Oscar.getClavesPublicas(),true);
29
30          Oscar.crearClaveConversacion(Juan.getL());
31          Alice.crearClaveConversacion(Juan.getL());
32
33          //comprobamos que Alice, Oscar y Juan tienen la misma clave
34          assertTrue(Alice.getClaveConversacion().isEqual(
35                  Oscar.getClaveConversacion()));
36          assertTrue(Alice.getClaveConversacion().isEqual(
37                  Juan.getClaveConversacion()));
38
39          //comrpobamos que Bob ahora tiene una clave diferente
40          assertTrue(!Alice.getClaveConversacion().isEqual(
41                  Bob.getClaveConversacion()));
42
43      }
```

Apéndice B

Serialización

En este apéndice se hablara de como convertir clases en conjuntos de bits y conjuntos de bits en clases. El propósito de esto es poder almacenar el estado de los objetos utilizados. De esta forma un usuario podrá almacenar un objeto, para que posteriormente otro usuario pueda recuperarlo y usarlo.

Para esto la clase y todos sus componentes deberán ser serializables.

Código B.1: Pasar una clase a un conjunto de bits

```
1   public byte[] toByteArray(Serializable o) {
2       ByteArrayOutputStream bos = new ByteArrayOutputStream();
3       ObjectOutput out = null;
4       byte[] byteRaw = null;
5
6       try {
7           out = new ObjectOutputStream(bos);
8           out.writeObject(o);
9           byteRaw = bos.toByteArray();
10
11          out.close();
12          bos.close();
13      } catch (IOException e) {
14          e.printStackTrace();
15      }
16      return byteRaw;
17  }
```

Código B.2: Convertir un conjunto de bits en una clase

```
1   public Serializable getSerializar(byte[] b) {
2       Serializable o = null;
3       try {
4           ByteArrayInputStream fis = new ByteArrayInputStream(b);
5
6           ObjectInputStream ois = new ObjectInputStream(fis);
7           o = (Serializable) ois.readObject();
8       } catch (ClassNotFoundException | IOException e) {
```

```
9                 e.printStackTrace();
10            }
11            return o;
12        }
```

Apéndice C

DropBoxAPI

Esta clase, que fue introducida en la Figura 3.3, supone un intermediario que facilita el trabajo con la core API de DropBox.

Código C.1: dropboxAPI.java

```
1  package com.ual.pfc.core.api;
2
3  import java.io.ByteArrayInputStream;
4  import java.io.ByteArrayOutputStream;
5  import java.io.File;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.IOException;
9  import java.io.InputStream;
10 import java.io.OutputStream;
11 import java.io.PrintWriter;
12 import java.util.ArrayList;
13 import java.util.Locale;
14 import java.util.Scanner;
15
16 import com.dropbox.core.DbxAppInfo;
17 import com.dropbox.core.DbxAuthFinish;
18 import com.dropbox.core.DbxClient;
19 import com.dropbox.core.DbxDelta;
20 import com.dropbox.core.DbxEntry;
21 import com.dropbox.core.DbxException;
22 import com.dropbox.core.DbxRequestConfig;
23 import com.dropbox.core.DbxWebAuthNoRedirect;
24 import com.dropbox.core.DbxWriteMode;
25
26
27 public class dropboxApi {
28     final String APP_KEY = "";
29     final String APP_SECRET = "";
30     String accessToken="";
31
32     private static Scanner input;
33     private static File tokensFile;
```

```
34      DbxClient client;
35
36      public dropboxApi() {
37          DbxRequestConfig config = new DbxRequestConfig(
38                  "JavaTutorial/1.0", Locale.getDefault().toString());
39          client = new DbxClient(config, accessToken);
40      }
41
42      public String upload(byte[] b, String path) {
43          ByteArrayInputStream inputStream = new ByteArrayInputStream(b);
44          String rutaSubida = "";
45
46          try {
47              DbxEntry.File uploadedFile = client.uploadFile(path,
48                  DbxWriteMode.add(), b.length, inputStream);
49
50
51              rutaSubida=uploadedFile.toString();
52
53          } catch (DbxException | IOException e) {
54              e.printStackTrace();
55          }
56
57          try {
58              inputStream.close();
59          } catch (IOException e) {
60              e.printStackTrace();
61          }
62
63          return rutaSubida;
64      }
65
66      public ArrayList<DbxEntry> listForder(String path){
67          ArrayList<DbxEntry> list= new ArrayList<DbxEntry>();
68          DbxEntry.WithChildren listing;
69          try {
70              listing = client.getMetadataWithChildren(path);
71              for (DbxEntry child : listing.children) {
72                  list.add(child);
73              }
74          } catch (DbxException e) {
75              e.printStackTrace();
76          }
77          return list;
78      }
79
80      public boolean existFoder(String path){
81          ArrayList<DbxEntry> list= new ArrayList<DbxEntry>();
82          DbxEntry.WithChildren listing;
83          try {
84              listing = client.getMetadataWithChildren(path);
85              System.out.println(listing.toString());
86              return true;
87
88          } catch (DbxException e) {
89              return false;
```

```
90          }
91      }
92
93      public boolean folderContain(String path,String newPath){
94          ArrayList<DbxEntry> list= new ArrayList<DbxEntry>();
95          DbxEntry.WithChildren listing;
96          try {
97              listing = client.getMetadataWithChildren(path);
98              for (DbxEntry child : listing.children) {
99                  list.add(child);
100             }
101         } catch (DbxException e) {
102             e.printStackTrace();
103         }
104         for(DbxEntry d: list){
105             if(d.name.compareTo(newPath)==0)
106             {
107                 return true;
108             }
109         }
110         return false;
111     }
112
113     public byte[] downloadFile(String path) {
114         ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
115         try {
116             DbxEntry.File downloadedFile = client.getFile(path, null,
117                     outputStream);
118
119         } catch (DbxException | IOException e) {
120             e.printStackTrace();
121         }
122
123         byte[] bytes=outputStream.toByteArray();
124         try {
125             outputStream.close();
126         } catch (IOException e) {
127             e.printStackTrace();
128         }
129
130         return bytes;
131     }
132
133     public DbxDelta<DbxEntry> getDelta(String cursor, String directorio) throws DbxException{
134         DbxDelta<DbxEntry> results=client.getDeltaWithPathPrefix(cursor, directorio);
135
136         return results;
137     }
138
139     public void deletePath(String path) {
140         try {
141             client.delete(path);
142         } catch (DbxException e) {
143             e.printStackTrace();
144         }
145     }
```

```
146
147     public void createFolder(String path) {
148         try {
149             client.createFolder(path);
150         } catch (DbxException e) {
151             e.printStackTrace();
152         }
153     }
154
155     public String update(byte[] b, String path) {
156         ByteArrayInputStream inputStream = new ByteArrayInputStream(b);
157         String rutaSubida = "";
158
159         try {
160             DbxEntry.File uploadedFile = client.uploadFile(path,
161                 DbxWriteMode.update(client.getFile(path, null,
162                         new ByteArrayOutputStream()).rev), b.length, inputStream);
163
164
165             rutaSubida=uploadedFile.name;
166
167         } catch (DbxException | IOException e) {
168             e.printStackTrace();
169         }
170
171         try {
172             inputStream.close();
173         } catch (IOException e) {
174             e.printStackTrace();
175         }
176
177
178         return rutaSubida;
179     }
180
181     public void deleteAllLess(String path, String doNotDeletePatron){
182         ArrayList<DbxEntry> files=listForder(path);
183         String aux;
184         for(DbxEntry d: files){
185             if(!d.name.contains(doNotDeletePatron)){
186                 aux=path+"/"+d.name;
187                 deletePath(aux);
188             }
189         }
190     }
191
192     public void updateMaxiveByteByPatron(String path, String patron, String patron2,byte[] b){
193         ArrayList<DbxEntry> files=listForder(path);
194         String aux;
195         for(DbxEntry d: files){
196             if(d.name.contains(patron)||d.name.contains(patron2)){
197                 aux=path+"/"+d.name;
198                 System.out.println(aux+"\n");
199                 update(b, aux);
200             }
201         }
```

```
202         }
203     }
```

Bibliografía

[1] J.-J. Climent, J. A. López-Ramos, P. R. Navarro, and L. Tortosa. Key agreement protocols for distributed secure multicast over the ring $e_p^{(m)}$. wit transactions on information and communication technologies, 45: 13–24 (2013).

[2] W. D. Diffie and M. E. Hellman. New directions in cryptography. ieeetransactions on information theory, 22(6): 644–654 (1976).

[3] DropBox. Datastore api documentation. `https://www.dropbox.com/developers/datastore/docs/js`.

[4] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. ieee transactions on information theory, 31(4): 469–472 (1985).

[5] Neal Koblitz. Elliptic curve cryptosystems. mathematics of computation 48, pp203–209(1987).

[6] V. Miller. Use of elliptic curves in cryptography(1985).

[7] National Institute of Standards and Technology (NIST). Specification for the advanced encryption standard (aes) ,2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[8] Oracle. Javatm cryptography architecture, api specification and reference. `http://docs.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.htm`.

[9] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. acm computing surveys, 35(3): 309–329 (2003).

[10] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. communications of the acm, 21(2): 120–126 (1978).

[11] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, siam j. computing 26, pp. 1484-1509 (1997).