**CLOUD-IN-ONE: INTEGRACIÓN LOCAL DE SERVICIOS DE ALMACENAMIENTO EN LA NUBE**

AUTOR

GUZMÁN PEDRAJAS, VÍCTOR

TUTOR

Saiz Noeda, Maximiliano

DEPARTAMENTO

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

CURSO:

2014 – 2015

Índice

[Introducción 1](#_Toc424681066)

[¿Qué es? 1](#_Toc424681067)

[Justificación 1](#_Toc424681068)

[Diferencias con otros servicios similares 2](#_Toc424681069)

[Uso de la aplicación 3](#_Toc424681070)

[Cio-crypt.exe 4](#_Toc424681071)

[Algoritmo de sincronización 5](#_Toc424681072)

[1. Obtención de cambios 5](#_Toc424681073)

[2. Reparación de colisiones 5](#_Toc424681074)

[3. Aplicación de cambios 7](#_Toc424681075)

[Estructura del código 7](#_Toc424681076)

[Patrones de diseño usados 8](#_Toc424681077)

[Encriptación 9](#_Toc424681078)

[Generación de la clave 9](#_Toc424681079)

[Encriptación y verificación 10](#_Toc424681080)

[Modificaciones en la librería 10](#_Toc424681081)

[¿Para qué sirve esta encriptación? ¿Ante qué protege? 10](#_Toc424681082)

[Debilidades - ¿Ante qué NO protege? 10](#_Toc424681083)

[Resultados y conclusiones 12](#_Toc424681084)

[Lenguaje de programación 12](#_Toc424681085)

[Código abierto 12](#_Toc424681086)

[Tests unitarios 12](#_Toc424681087)

[Objetivos del proyecto 12](#_Toc424681088)

[Posibles ampliaciones 13](#_Toc424681089)

[Bibliografía y enlaces de interés 14](#_Toc424681090)

# Introducción

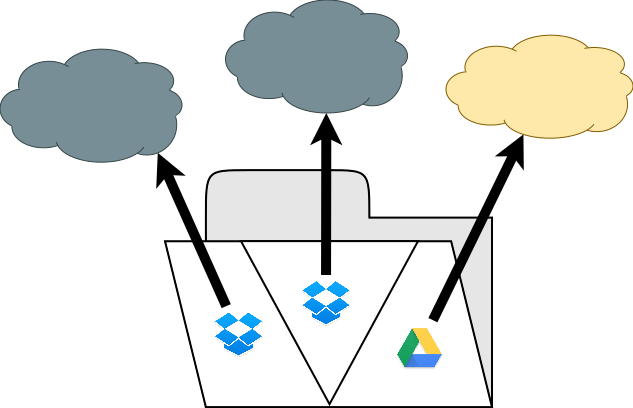
## ¿Qué es?

CLOUD-IN-ONE es un proyecto orientado a proporcionar una interfaz transparente y segura de servicios de almacenamiento en la nube.

Esto se consigue mediante una aplicación que actualiza en segundo plano los archivos de la carpeta de sincronización con todas las cuentas vinculadas al sistema. El proceso es similar al cliente oficial de Dropbox, pero conectado a diferentes cuentas. Además, todos los ficheros de los diferentes servicios de almacenamiento se agregan en una misma carpeta de forma transparente al usuario, por lo que el usuario no necesita saber a qué servicio está subiendo sus ficheros, solo necesita saber que se han almacenado de forma segura.

El sistema puede, por otro lado, encriptar los ficheros antes de subirlos a los respectivos servicios. Así, si alguien accediera a la cuenta remota (por ejemplo, desde la web de Dropbox) no podría leer ninguno de los archivos.

Es necesario recalcar que el propósito de CLOUD-IN-ONE es ayudar a organizar fácilmente los ficheros que se encuentran repartidos entre varias cuentas remotas. El propósito no es evitar los límites de espacio impuestos por los servicios de almacenamiento remoto.



## Justificación

A lo largo de los últimos años, han aparecido multitud de servicios de almacenamiento en la nube, habitualmente con un límite de espacio disponible. Esto hace que, aunque en fragmentos pequeños, dispongamos de una gran cantidad de espacio de almacenamiento.

Este espacio está repartido en varios servicios, así que tenemos que decidir dónde queremos almacenar nuestra información. Sin embargo, el usuario medio sólo quiere almacenar su información, pero no necesita que ésta esté guardada en una cuenta o un servicio concretos.

Por otro lado, aunque existen herramientas comerciales que solucionan este problema de diferentes maneras, ninguna de ellas tiene una base de código abierto.

Otro aspecto importante del proyecto es la posibilidad de encriptar los ficheros que se envían a los servidores remotos. Muchos servicios de almacenamiento no ofrecen ninguna garantía de privacidad: cualquiera con acceso a los datos, puede leerlos. Esto plantea riesgos en caso de que la empresa que presta el servicio de almacenamiento tenga un fallo de seguridad, o permita el acceso a sus sistemas a un tercero.

El objetivo del proyecto es crear un sistema que agregue varios servicios de almacenamiento a una carpeta local. De esta manera, para utilizar la aplicación, el usuario solo tiene que guardar un fichero en una carpeta. El sistema se encargará de actualizar las cuentas remotas y la carpeta local en segundo plano.

## Diferencias con otros servicios similares

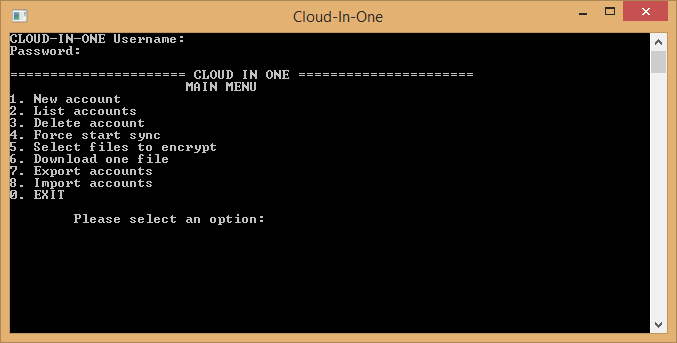
Existen otros servicios similares a CLOUD-IN-ONE, pero hay varias diferencias importantes:

* Pese a que la aplicación está inspirada en la forma de trabajar de la aplicación oficial de Dropbox, CLOUD-IN-ONE permite vincular varias cuentas al mismo tiempo, mientras que la aplicación de Dropbox sólo permite una.
* CLOUD-IN-ONE se integra con el sistema operativo: el usuario solo se tiene que preocupar de guardar sus ficheros en una carpeta local, y la aplicación se encargará de monitorizarla y subirla a un servicio remoto. Otras aplicaciones como Jolicloud, CloudKafé o MultCloud se basan en web, con lo que el usuario se ve obligado a trabajar desde el navegador y no desde el propio sistema.
* Servicios como *Gladinet*, *odrive* u *otixo* separan los servicios en distintas carpetas, CLOUD-IN-ONE agrega todas las cuentas en una misma carpeta. De esta manera el usuario guarda sus ficheros en la carpeta, y CLOUD-IN-ONE organizará los ficheros entre los servicios.
* CLOUD-IN-ONE permite encriptar los ficheros para que nadie que no esté autorizado pueda leerlos.

# Uso de la aplicación

Al iniciar la aplicación, el usuario debe introducir un usuario y una contraseña, que se usarán para autenticarlo y para la encriptación.

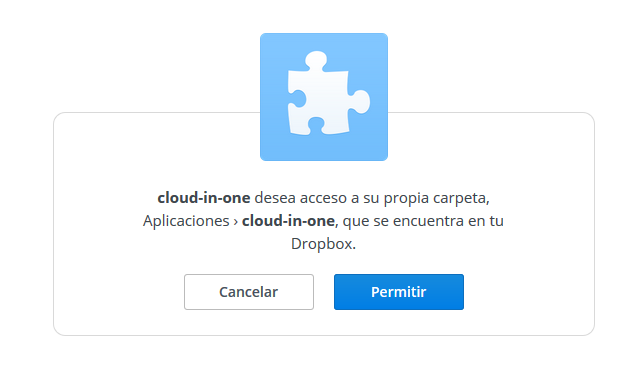
Lo siguiente que verá el usuario será el menú principal:



Aquí el usuario puede elegir una de 8 opciones:

1. New account

Esta opción permite añadir una nueva cuenta de Dropbox. Al elegirla se pedirá un nombre de cuenta y se abrirá una ventana de navegador para autorizar el acceso a la cuenta.



1. List accounts

Permite mostrar la lista de cuentas añadidas al sistema.

1. Delete account

Permite eliminar una cuenta del sistema. Los archivos vinculados a la cuenta, que estén encriptados, se desencriptarán y se subirán a la cuenta correspondiente. Tras esto se eliminarán de la carpeta de sincronización local.

1. Force start sync

Lanza el proceso de sincronización. Si no se pulsa esta opción, se lanzará automáticamente cada 5 minutos.

1. Select files to encrypt

Muestra una lista de los ficheros almacenados en la carpeta local, y permite seleccionar cuál queremos marcar para encriptar. Una vez marcado, la siguiente vez que este archivo se suba al servicio remoto, se encriptará antes.

1. Download one file

Permite descargar un archivo que esté almacenado en nuestra cuenta remota a nuestra carpeta de sincronización.

1. Export accounts

Una vez seleccionada, pide una ruta al usuario y exporta un fichero encriptado con la información de las cuentas añadidas actualmente a la aplicación.

1. Import accounts

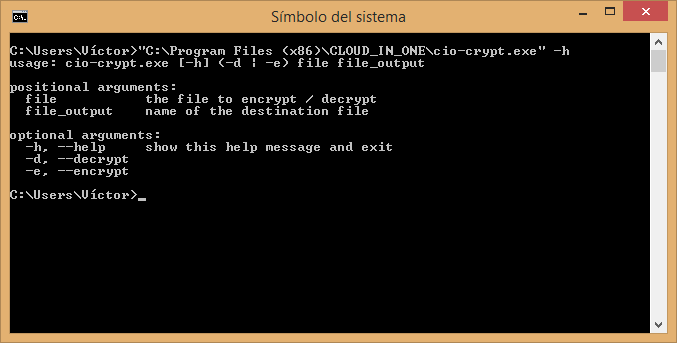
Pide un fichero exportado mediante la opción anterior. Una vez proporcionado, añade las cuentas a la aplicación de la misma forma que estaban cuando se exportó.

## Cio-crypt.exe

Junto con la aplicación principal se distribuye un ejecutable separado, *cio-crypt.exe*. Este ejecutable se encarga únicamente de encriptar/desencriptar un fichero concreto.

Esto puede usarse para el acceso directo desde la web de Dropbox a los ficheros encriptados. Por ejemplo, necesitemos acceso a un fichero encriptado, pero lo hemos descargado desde Dropbox directamente. O queremos almacenar un fichero desde la web de Dropbox, pero queremos que esté encriptado.

Cio-crypt es una aplicación puramente de línea de comandos:



Así los posibles usos serían:

* $ “C:\Program Files (x86)\CLOUD\_IN\_ONE\cio-crypt.exe” --encrypt <fichero original> <fichero de salida>

La aplicación pedirá un usuario y una contraseña para realizar la encriptación. Tras esto, el fichero salida tendrá un contenido como:

“sc\_\_$€…ã\_râ4,¤Qÿ|6´¦q\_d<\_ä¿\_¥~®ò”

* $ “C:\Program Files (x86)\CLOUD\_IN\_ONE\cio-crypt.exe” --decrypt <fichero encriptado> <fichero de salida>

La aplicación pedirá un usuario y una contraseña para realizar la desencriptación.

# Algoritmo de sincronización

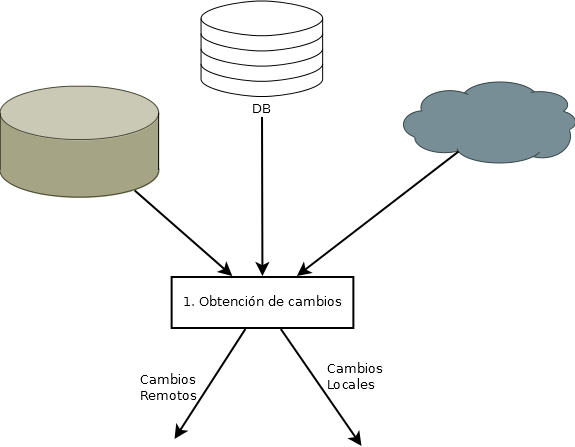
El núcleo principal de CLOUD-IN-ONE es el algoritmo de sincronización. Éste se encarga de obtener los cambios de todas las cuentas remotas vinculadas al sistema, compararlos con los cambios locales, y dejar un estado consistente en todas ellas.

El algoritmo tiene tres fases:

## Obtención de cambios

El algoritmo hace una petición de cambios a todas las cuentas remotas, y analiza la carpeta de sincronización en busca de cambios en los ficheros locales.

Las cuentas remotas devuelven, a través de sus respectivas APIs, una lista de creaciones, modificaciones y borrados de ficheros. A su vez, recorre la carpeta de sincronización local y, para cada fichero, calcula un hash y lo compara con el hash almacenado anteriormente. Si son diferentes se añade el archivo a la lista de cambios locales.



## Reparación de colisiones

Una vez recibidos los cambios remotos y locales, el algoritmo los analiza para evitar que cambios que se refieran a un mismo archivo puedan sobrescribirse entre ellos.

Por ejemplo: un archivo, subido a la cuenta A, ha sido borrado en el servidor remoto; mientras que el mismo archivo, en la copia local, ha sido modificado. En este caso, al analizar los cambios, el sistema detectaría que, para evitar pérdidas de datos, el archivo no debe borrarse, sino que debe ser la modificación la que prevalezca.

Cuando el algoritmo no es capaz de resolver un conflicto automáticamente, crea una copia conflictiva para que el usuario pueda decidir qué versión quiere mantener.

La siguiente tabla muestra los posibles casos a la hora de detectar conflictos entre cambios de un mismo fichero.

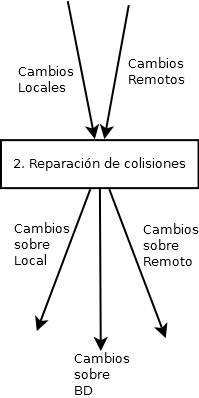
* Los casos sin color son aquellos que el algoritmo puede decidir por sí mismo, ya que su única acción posible es eliminar uno de los dos cambios. En estos casos, el algoritmo siempre decidirá quedarse con la acción menos destructiva (lo que no sea un borrado)
* Los casos en amarillo indican que el algoritmo crea una copia conflictiva para que el usuario decida.
* El caso en naranja indica que el algoritmo evalúa si la modificación en los dos cambios es exactamente la misma. Si es así, el algoritmo considerará solo una de ellas; si no, creará una copia conflictiva como en el caso anterior.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Creación | Modificación | Borrado |
| Creación | Conflicto | Conflicto | Creación |
| Modificación | Conflicto | Conflicto / Modificación | Modificación |
| Borrado | Creación | Modificación | Borrado |

*Casos posibles en el algoritmo de detección de conflictos*

Hay dos tipos de copia conflictiva: entre un archivo local y un archivo remoto, y entre dos remotos al mismo tiempo. Estos dos tipos funcionan de la misma manera, pero el nombre del archivo conflictivo será diferente.

* Local ↔ Remoto. Es el caso clásico de la aplicación de Dropbox. Si se modifica el mismo archivo en la carpeta local y en el servidor remoto, CLOUD-IN-ONE creará una copia con el nombre “<archivo>\_\_CONFLICTED\_COPY\_\_<fecha>”.
* Remoto ↔ Remoto. Este caso es único de CLOUD-IN-ONE. El caso más común es crear un archivo, con el mismo nombre, de forma remota en dos servidores al mismo tiempo. Al sincronizar, CLOUD-IN-ONE creará una copia con el nombre “<archivo>\_\_CONFLICTED\_COPY\_\_FROM\_<cuenta>\_<fecha>”, donde cuenta es el nombre que se le ha dado a la cuenta remota en el sistema.



## Aplicación de cambios

Finalmente, el algoritmo aplica los resultados del paso anterior. Esto implica enviar los cambios definitivos a cada uno de los servidores remotos, encriptándolos si hace falta; escribirlos a la carpeta de sincronización local y almacenarlos en la base de datos.

* Aplicación en las cuentas remotas.

Al enviar los cambios a las cuentas remotas, es posible que el fichero no quepa en alguna de ellas. El algoritmo recorrerá las cuentas hasta encontrar una en la que el fichero quepa, o devolverá un error si no cabe en ninguna.

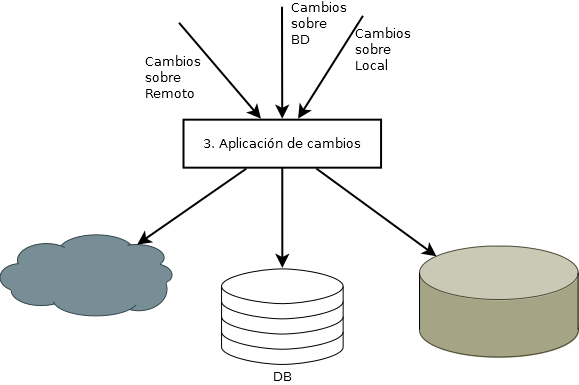
Además, si el fichero está marcado para encriptar (o si el fichero estaba encriptado en el servidor remoto originalmente), antes de subir el archivo éste será encriptado.

* Aplicación en la carpeta local.

Al aplicar los cambios en la carpeta local, el caso más común es tener que descargar un fichero. En ese caso, el algoritmo intentará desencriptar el fichero remoto, por si acaso ha sido encriptado de forma separada. Si la desencriptación falla, se usará el fichero directamente.

* Base de datos

Esta fase almacena en la base de datos el estado final después de haber aplicado los cambios.



# Estructura del código

CLOUD-IN-ONE se ha programado en Python usando un paradigma orientado a objetos.

El siguiente diagrama representa las clases de la aplicación:

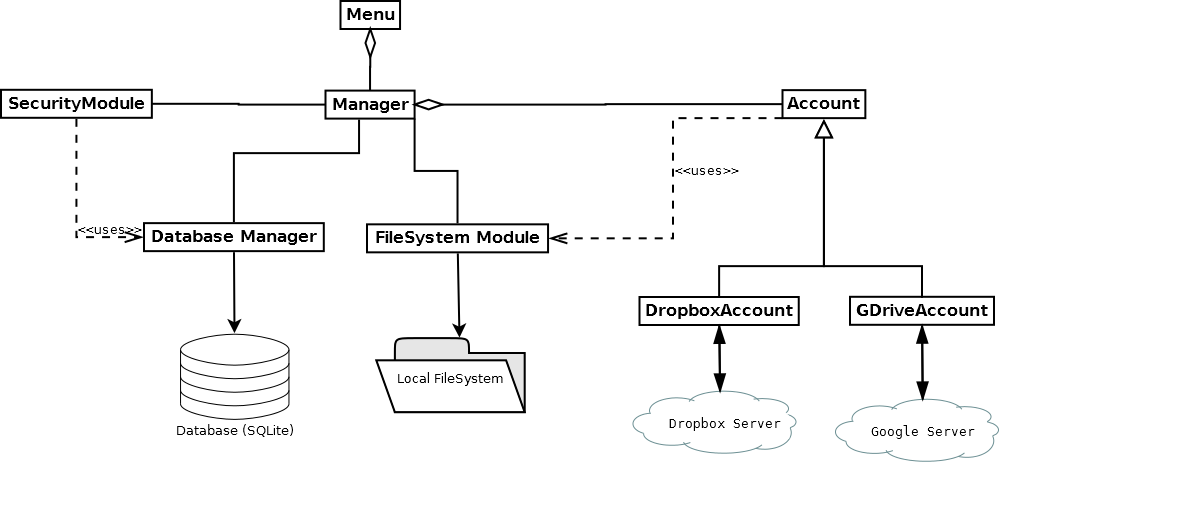


Ilustración 1 - Diagrama UML general

Se puede ver que la clase Manager está relacionada con prácticamente todas las demás del código. Esto es así porque esta es la clase encargada de ejecutar el algoritmo de detección y aplicación de cambios. Esta clase se encarga también de gestionar la lista de cuentas, además de crear e inicializar otras clases, como DatabaseManager, SecurityModule o FileSystemModule.

DatabaseManager es una clase que engloba todas las funciones relacionadas con la Base de Datos.

SecurityModule es la clase encargada de encriptar y desencriptar los ficheros, y de autenticar al usuario.

En este diagrama se ha omitido la clase *Logger*, que se encarga de gestionar la escritura de un fichero de Log tanto por consola como a un fichero. Esta clase está vinculada a todas las demás clases, que crean una instancia en su constructor.

## Patrones de diseño usados

La sección que controla el acceso a las cuentas implementa un patrón **Strategy**, lo que permite añadir fácilmente nuevos servidores remotos, haciendo que implementen la interfaz *Account*.

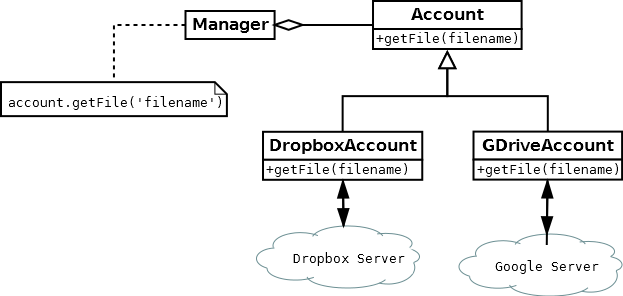


Ilustración 2 - Ejemplo del patrón Strategy con el método getFile()

A su vez, las clases que llaman a los servidores remotos implementan un patrón **Façade**, que simplifica el uso de las APIs remotas por parte del resto de la aplicación.

El patrón **Façade** también se utiliza en el resto de clases para simplificar las llamadas a las librerías, la base de datos y el sistema de ficheros.

# Encriptación

CLOUD-IN-ONE utiliza un sistema de encriptación simétrica con una clave generada en base al usuario y contraseña introducidos durante el registro.

Para la encriptación en sí se utiliza una librería de código abierto llamada *simple-crypt*, que a su vez utiliza la librería de encriptación en Python más usada: *PyCrypto.* Sin embargo, *simple-crypt* tenía ciertas limitaciones, por lo que me vi obligado a hacer ciertas modificaciones en la librería.

## Generación de la clave

La clave usada para la encriptación se genera mediante una función hash SHA256. Esta función recibe como parámetro una cadena que consiste en:

* Una cadena llamada “sal”. Que en cierta manera protege ante métodos de fuerza bruta precalculados como tablas arcoíris.

Típicamente se recomienda que esta sal se cree aleatoriamente durante cada registro, pero dado que el registro se realiza en local, y necesitamos obtener la misma clave para una misma combinación de usuario-contraseña, fue necesario utilizar una sal constante escrita en el código.

* El nombre de usuario elegido durante el registro.
* La contraseña elegida durante el registro.



La función genera una clave de 256bits que no es posible asociar de ninguna manera a la combinación de usuario y contraseña originales.

La librería *simple-crypt*, por su parte, en base a la clave que la aplicación envía, regenera las claves que se usarán para el cifrado real, usando PBKDF2, SHA256 y 100.000 iteraciones.

## Encriptación y verificación

El proceso de encriptación en sí se deja a la librería *simple-crypt*, que utiliza un cifrado AES256 en modo CTR.

El mensaje cifrado incluye un código de verificación HMAC, lo que permite comprobar que un mensaje desencriptado corresponde con el mensaje original, y no ha sido modificado en la transacción.

Por último, la librería añade un encabezado que permite identificar un mensaje como encriptado, y con qué versión de la librería se ha hecho.

Así, el mensaje final que se envía se compone del encabezado, sal generada por la librería, datos encriptados y HMAC. Esto significa que el tamaño del mensaje final será mayor una vez encriptado, pero lo será por un factor constante.

## Modificaciones en la librería

La librería *simple-crypt* solo está pensada para encriptar cadenas de texto de tamaño relativamente pequeño, por lo que CLOUD-IN-ONE se distribuye con una versión modificada para permitir encriptar textos mayores que el espacio disponible en RAM.

Esto se consigue leyendo el fichero por bloques que caben en memoria, y cifrándolos secuencialmente. Así, cada bloque será como un mensaje cifrado separado, con su encabezado y su código de verificación.

Siguiendo una filosofía de software libre, estos cambios se mandaron al autor de la librería pero, después de analizarlos y considerar el caso de uso, decidió que la librería debe seguir siendo simple de usar y no merecía la pena integrar ese cambio. Además, el autor consideró que el cambio podía plantear un problema de seguridad, ya que al ser bloques independientes, un atacante podría, por ejemplo, eliminar uno de ellos y el usuario no se enteraría.

## ¿Para qué sirve esta encriptación? ¿Ante qué protege?

La encriptación en la aplicación CLOUD-IN-ONE permite evitar que nadie que no seamos nosotros pueda leer los datos que están en un servicio remoto – y, por tanto, fuera de nuestro control –. Esto significa que si nuestros datos están subidos a, por ejemplo, Dropbox y un empleado de la empresa decide leer nuestra información, gracias a la encriptación, no podría.

Este tipo de encriptación también protege ante atacantes que obtengan nuestra información de acceso a la cuenta remota. Si algún atacante obtiene nuestro usuario y contraseña de Dropbox, no podrá leer la información que hay allí almacenada.

Por último, dado que la encriptación se realiza en local, incluso si el atacante consigue llevar a cabo un ataque de man-in-the-middle, suplantando la identidad del servidor remoto, el atacante no podrá leer esta información.

## Debilidades - ¿Ante qué NO protege?

El sistema de encriptación de CLOUD-IN-ONE no puede proteger de todas las situaciones en las que un atacante podría acceder a nuestra información.

Por ejemplo, si un atacante tiene acceso a nuestras credenciales del servidor remoto, aunque no pueda leer los archivos encriptados, sí podría eliminarlos o sustituirlos por otros.

Otro punto flaco del sistema de encriptación es el ordenador local. Dado que la carpeta de sincronización está totalmente desencriptada para el uso del usuario, alguien con acceso a esta carpeta podría acceder a los ficheros sin ningún problema.

Además, la base de datos se encuentra almacenada en un fichero en el ordenador local, por lo que cualquiera con acceso físico a este fichero podría modificar datos clave de la aplicación, como la cuenta asociada a los ficheros.

# Resultados y conclusiones

En un principio el proyecto iba a ser mucho más ambicioso, con más funcionalidades y más servicios de almacenamiento disponibles. Sin embargo, no ha sido posible en el tiempo disponible, y hubo que reducir el ámbito del mismo.

## Lenguaje de programación

La aplicación se ha programado en Python 3.4.2 (posteriormente actualizada a 3.4.3). Considero que esto fue un gran acierto, ya que aunque no era el lenguaje en el que más experiencia tenía, se trata de un lenguaje muy flexible y sencillo, que me ha permitido avanzar muy rápidamente; y, por otro lado, era un lenguaje que tenía mucho interés en aprender más a fondo.

Además, Python dispone de intérpretes en la mayoría de las plataformas, haciendo que la aplicación sea multiplataforma con muy poco esfuerzo.

## Código abierto

Uno de los objetivos del proyecto es crear una alternativa de código abierto a aplicaciones comerciales similares. Esto se ha hecho mediante un repositorio en mi cuenta GitHub, con una de las licencias que recomendaba GitHub inicialmente, la licencia MIT, que prácticamente permite a cualquiera hacer cualquier uso del código.

Lamentablemente la aplicación no ha atraído la atención de nadie con ánimo de colaborar, pero gracias a ella yo sí he colaborado en algún otro proyecto de código abierto.

## Tests unitarios

Desde el principio del proyecto decidí hacer tests unitarios de las funcionalidades principales de la aplicación. Aunque la batería de tests no es exhaustiva y completa, sí que me ha permitido solucionar algunos errores que habrían sido muy difíciles de encontrar si no la hubiera tenido.

Otra gran ventaja de escribir tests unitarios ha sido forzarme a pensar seriamente cuál era el comportamiento deseado de algunas secciones del código, obligándome de esta forma a mejorar el diseño.

Por otro lado, incorporé un sistema de integración continua (Travis CI) que me permitió probar la aplicación cada vez que hacía algún cambio en el repositorio GitHub. Además funcionaba sobre un sistema Linux, con lo que se probaba la multiplataforma al mismo tiempo.

## Objetivos del proyecto

☑ Programación en Python

☑ Integración en el sistema operativo

☑ Uso de APIs de terceros

☑ Multiplataforma

☑ Código abierto

☑ Sincronización con varias cuentas remotas al mismo tiempo

☑ Encriptación

☑ Uso de patrones de diseño

☑ Uso de tests unitarios

☐ Integración con varios servicios de almacenamiento.

Aunque la intención inicial era disponer de varios servicios de almacenamiento remoto, finalmente hubo que reducirlo únicamente a Dropbox. Sin embargo, gracias a la arquitectura de la aplicación (y, en concreto, al patrón Strategy usado para la sección de cuentas remotas), añadir un nuevo tipo de cuenta no es complicado, y queda para una posible ampliación futura.

☒ Facilidad de uso.

Aunque la aplicación estaba pensada para el público general, desde el principio deseché la idea de crear una interfaz gráfica, por lo menos hasta tener el proyecto mucho más avanzado. Cuando aparecieron funcionalidades que necesitaban interacción directa con el usuario, era demasiado tarde para hacer una interfaz gráfica desde cero, y tuve que hacer un apaño con un menú desde línea de comandos, que no es en absoluto fácil de usar para el público general.

## Posibles ampliaciones

Debido a la reducción del ámbito del proyecto, hubo que dejar fuera algunas funcionalidades que inicialmente estaban planeadas y habrían sido útiles. Quedan como ampliaciones futuras:

* Integración con otros servicios de almacenamiento.
* Posibilidad de ignorar archivos.
* Interfaz gráfica para autorizar el acceso a las cuentas.
* Mejorar la eficiencia de la función de descubrimiento de cambios locales.
* Añadir iconos en el explorador de ficheros del sistema indicando la cuenta vinculada.

# Bibliografía y enlaces de interés

Blablabla

<http://victorlin.me/posts/2012/08/26/good-logging-practice-in-python>