Seguridad en servicios de almacenamiento
Análisis de Dropbox y Mega

INSTITUTO NACIONAL DE CIBERSEGURIDAD

SPANISH NATIONAL CYBERSECURITY INSTITUTE

incibe_

Autor

Jesús Díaz Vico

Este estudio ha sido elaborado con la colaboración de David Cantón Araujo, Daniel Fírvida Pereira, Asier Martínez Retenaga y Antonio Rodríguez Fernández.

INDICE

1 Sobre este estudio

Los servicios «en la nube» son cada vez más populares, recibiendo un número creciente de usuarios. Entre los usos más comunes, se encuentran las aplicaciones para almacenamiento en la nube. El presente estudio recoge diferentes aspectos relacionados con estos servicios y la seguridad que integran. Atendiendo a su estructura, se ha dividido en dos grandes bloques:

- En la primera parte, hasta la sección 3 inclusive, se han introducido los principales aspectos que pueden afectar a la seguridad de soluciones de almacenamiento en la nube, principalmente desde una perspectiva técnica, pero también mencionando otros aspectos de importancia. En concreto, se ha puesto énfasis en los métodos de cifrado y autenticación, las políticas de deduplicación y la compartición de datos entre usuarios. Para estos puntos, se han descrito las principales alternativas y las implicaciones que cada una de ellas pueden tener sobre la seguridad y privacidad de los usuarios.
- En la segunda parte del estudio, a partir de la sección 4, se establece un marco común para analizar aplicaciones de almacenamiento en la nube. Esto se emplea en las secciones 5 y 6 analizado la seguridad de dos de las aplicaciones de uso mayoritario en el mercado, disponibles para cualquier usuario. Las aplicaciones analizadas, Dropbox[1] y Mega[2], son representativas en cuanto a que la primera es probablemente la más conocida, y la segunda, además de ser una aplicación popular, pone un especial énfasis en la seguridad. Se ha comprobado qué medidas incorporan estas dos aplicaciones para los aspectos descritos en la primera parte del estudio. En cualquier caso, es importante tener en cuenta que las pruebas[3] realizadas durante este estudio tuvieron lugar en el último trimestre de 2014, por lo que, conforme pase el tiempo, es probable que surjan aspectos que difieran de lo que aquí se describe.

Con los principios expuestos en la primera parte del estudio, es posible evaluar qué soluciones se adaptan mejor a las necesidades específicas que puedan presentarse. El análisis incluido en la segunda parte, además de destinarse a dos de las aplicaciones más populares, puede servir como marco base para reproducirlo a otras aplicaciones o extenderlo conforme a requisitos específicos.

[1] https://www.dropbox.com/
[2] https://mega.co.nz/
[3] Todas las pruebas realizadas durante este estudio se llevaron a cabo con cuentas de prueba, creadas específicamente para tal fin.

2 Introducción

El concepto de una nube de servicios electrónicos globales empezó a fraguarse con investigadores como Joseph C.R. Licklider, que ya en 1963, en un memorándum para ARPA[4], contempló las ventajas que proporcionaría una infraestructura de red que permitiese compartir servicios[5]. Más aún, predijo que una de las mayores dificultades que tal infraestructura encontraría sería la compatibilidad entre diferentes lenguajes de programación y protocolos de comunicaciones.

Esta idea evolucionó en la ARPANet, que a su vez dio origen a lo que hoy se conoce como Internet. A medida que Internet ha ido llegando a todos los ámbitos de empresas y personas, la tipología de los productos y servicios diseñados para compartir información ha evolucionado de forma acorde a las necesidades de los nuevos usuarios. Así, no sólo se ofrecen ya productos como el programa para *grid-plotting* al que se refería Licklider, sino utilidades de uso común como almacenamiento, servidores web, sistemas de correo electrónico, etc.

La evolución y constante mejora de la tecnología disponible, ha influido notablemente en este aspecto: el aumento del ancho de banda y el desarrollo de estándares internacionales para la interoperabilidad de distintos dispositivos, han facilitado la provisión de servicios desde lo que se ha venido a llamar **la nube**.

Estos hechos están provocando que en la actualidad se estén derivando a la nube muchas actividades que hasta hace pocos años se hacían «en local» o en infraestructuras corporativas privadas, como escuchar música (p.e. Spotify), ver vídeos (p.e. Youtube), correo electrónico (p.e. Gmail), almacenamiento (p.e. Dropbox), etc.

Estos servicios propician una mejor experiencia al usuario, ya que evitan tener que llevar a cabo costosas o complicadas tareas de instalación, configuración y mantenimiento, permitiendo a los usuarios preocuparse exclusivamente de sus tareas cotidianas o profesionales, incurriendo también en menores costes en muchas ocasiones. Además, el usuario final también recibe importantes ventajas, como una mayor disponibilidad del servicio.

Como toda ciencia o tecnología, con el tiempo, han madurado conceptos para ayudar a comprender y mejorar este nuevo entorno. Así, como se explica en «*The NIST Definition of Cloud Computing*» [1], dentro de los modelos de servicios ofertados dentro de la nube se puede hablar de ***Software as a Service*** **(SaaS**, Software como Servicio**)**, ***Platform as a Service*** **(PaaS**, Plataforma como Servicio**)** e ***Infrastructure as a Service*** **(IaaS**, Infraestructura como Servicio**)**, que pueden ser utilizados a través de clientes de escritorio o de clientes web.

En concreto, SaaS se refiere a la utilización de software disponible desde la nube (p.e.

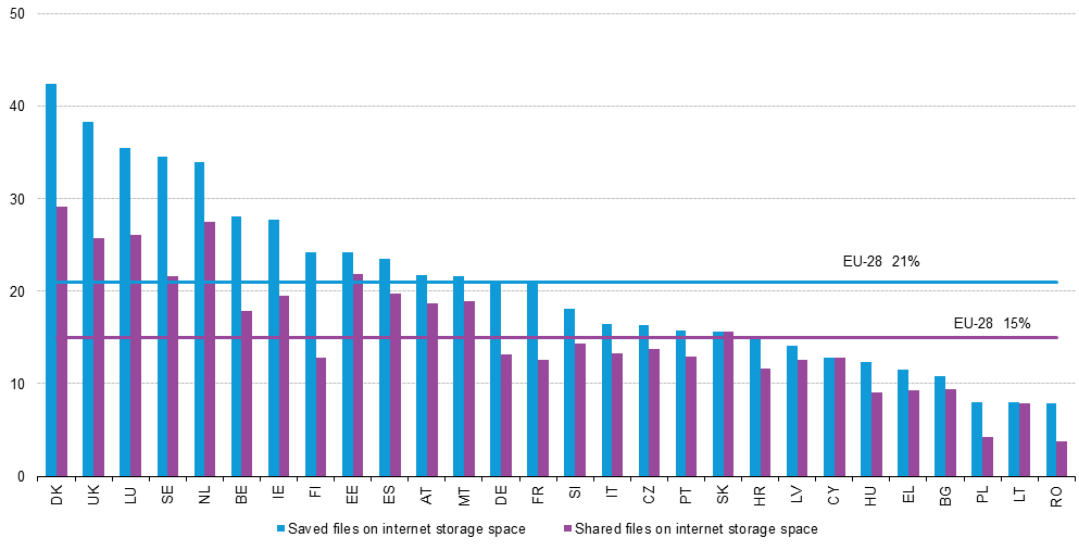
[4] http://es.wikipedia.org/wiki/Defense_Advanced_Research_Projects_Agency.
[5] En concreto, ejemplifica su idea a partir de un programa de *grid-plotting* que pudiera compartirse en una red de 4-8 ordenadores. Fuente: http://www.kurzweilai.net/memorandum-for-members-and-affiliates-of-the-intergalactic-computer-network.

Spotify, Youtube, Gmail, Dropbox); PaaS a la utilización de una plataforma de cómputo de forma remota (p.e. sistemas operativos configurados con entornos de desarrollo, bases de datos, servidores web); e IaaS al aprovechamiento de infraestructuras deslocalizadas (p.e. máquinas virtuales y sistemas de almacenamiento). Nótese que incluso es posible establecer inter-relaciones entre distintos proveedores de servicios. De esta manera, por ejemplo, un proveedor de IaaS puede ofrecer máquinas virtuales a un proveedor de PaaS que montase servidores XAMP que podrían ser a su vez contratados por una emisora de televisión online via web (es decir, un proveedor de SaaS). Evidentemente, no siempre tienen que estar presentes los tres eslabones en una misma cadena.

Uno de los servicios más populares de los últimos años es sin duda el de **almacenamiento en la nube**. En el presente estudio se estudia la seguridad de dos de las aplicaciones más populares: Dropbox, quizá la más conocida y extendida de todas, y Mega, también bastante popular y con un énfasis en la seguridad desde sus inicios.

Como se muestra en la Figura 1, en media, el 21% de los ciudadanos de la Unión Europea (EU-28) utilizó algún sistema de almacenamiento en la nube durante 2014 para guardar ficheros, y un 15% los utilizó para compartir. En algunos países (Dinamarca, Reino Unido, Luxemburgo, Suecia y Holanda), superándose el 30% y el 20%, respectivamente (en algunos casos, con creces).

Figura 1. Uso de sistemas de almacenamiento en la nube en países de la Unión Europea.



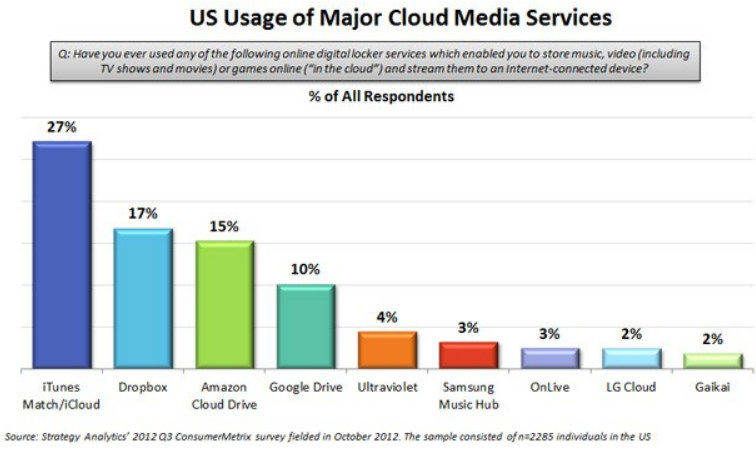
Fuente: EuroStat[6]

[6] http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals#Publications.

Probablemente, Dropbox sea la primera aplicación de este tipo en expandirse globalmente, publicaendo su primera versión en 2008. Desde entonces, han surgido numerosas alternativas con mayor o menor éxito, habiendo hoy en día varias decenas disponibles[7].

La mayor parte de la cuota de mercado está repartida actualmente entre unos pocos servicios, tal y como se desprende del estudio de mercado realizado por *Strategy Analytics* en el tercer cuatrimestre de 2013, donde preguntaron a 2300 ciudadanos estadounidenses (ver Figura 2). Como se puede observar, iCloud, Dropbox y Amazon Cloud Service sumaban a finales de 2013 aproximadamente el 50% de la cuota de mercado, seguidos por Google Drive con un 10% de cuota.

Figura 2. Cuota de mercado de las aplicaciones de almacenamiento en la nube, en EEUU. Estudio realizado en 2013 por Strategy Analytics.



Fuente: Strategy Analytics[8]

Asimismo, se pueden encontrar otras aplicaciones que priman la seguridad aún a costa de, quizás, sacrificar eficiencia en alguna otra característica. Entre las opciones de este tipo, las más conocidas probablemente sean Mega, SpiderOak[9] y Wuala[10].

El objetivo del presente documento es analizar las características de seguridad y privacidad de Dropbox y Mega, dos de las principales aplicaciones de almacenamiento

[7] Ver por ejemplo http://thesimplecomputer.info/behind-the-curtain-of-encrypted-cloud-storage para un listado bastante completo.
[8] http://www.engadget.com/2013/03/21/strategy-analytics-cloud-media-market-share/
[9] https://spideroak.com/
[10] http://www.wuala.com/

en la nube disponibles gratuitamente en el mercado. En el documento, antes de su análisis se exponen los principales conceptos que influirán en el análisis y que se utilizan en este estudio como base.

3 Aspectos a tener en cuenta

Con el fin alcanzar una buena base que permita comprender los factores que afectan a la seguridad de un sistema de almacenamiento en la nube, es necesario saber cómo funcionan. A continuación se resumen los principales aspectos que afectan al grado de privacidad ofrecida a los usuarios finales.

3.1 Gestión y utilización de cifrado

Una de las piedras angulares de la seguridad de toda funcionalidad criptográfica es el proceso de generación de claves, como ya observó Kerckhoffs en 1883 [2]. En este sentido, es aconsejable adherirse a recomendaciones nacionales o internacionales como la *Special Publication 800-133* del NIST [3]. Pero no es sólo importante el algoritmo de derivación o generación de claves, también es importante dónde se generan y cómo se gestionan después.

Hay aplicaciones que generan claves para cifrado en los servidores, pero que para enviar la información utilizan otra clave e incluso otro tipo de cifrado (normalmente, para las comunicaciones se recurre a SSL/TLS). De esta forma, aunque los datos se transmitan cifrados, el servidor dispondrá de acceso a ellos, por lo que las garantías con respecto a la privacidad del usuario disminuyen.

Algunas aplicaciones también permiten al usuario generar sus claves utilizando para ello herramientas externas, para posteriormente importarlas con el cliente de la aplicación de almacenamiento en la nube. En este aspecto, siempre y cuando la herramienta externa también siga las recomendaciones de generación de claves, ésta es la opción más robusta, ya que permite al usuario escoger herramientas que han sido diseñadas específicamente para tal fin y en las que el usuario ya confíe (por ejemplo, por experiencia personal anterior).

Obviamente, la gestión del cifrado de datos es un aspecto fundamental para garantizar la confidencialidad de los mismos. En este aspecto, hay varios puntos a tener en cuenta:

1. El **algoritmo de cifrado** en sí mismo: de nuevo, el uso de algoritmos verificados por la comunidad científica y tecnológica es esencial. Estándares como AES [4] o IDEA [5] suelen ser buenas alternativas adoptadas por muchos proveedores.
2. De la misma forma que dónde se generan las claves criptográficas, es importante **dónde** (cliente o servidor), **y para qué** (transmisión o almacenamiento), **se cifran los datos**. Por ejemplo, si los datos se cifran únicamente durante su transmisión, pero luego son almacenados en claro en los servidores, sigue existiendo una amenaza para la privacidad de los usuarios.

3. **Con qué clave se cifran los datos**: si es con una clave creada y gestionada por el proveedor del servicio, el usuario «pierde» el control efectivo sobre sus datos una vez son transmitidos a los servidores. Si se hace con una clave creada por el usuario (y si es con una aplicación externa, mejor), el usuario seguirá controlando los datos aún cuando estos hayan sido enviados al servidor.

3.2 Autenticación

Como en toda comunicación, es importante que ambos extremos se autentiquen entre sí, así como a la información que transmiten. Por un lado, **los servidores deberían autenticarse siempre** antes de iniciar una comunicación, utilizando certificados digitales emitidos por autoridades confiables. Como se observa en la definición del protocolo TLS [6, p. 90], mientras al menos el servidor esté autenticado, la transmisión será segura frente a ataques del tipo *man-in-the-middle*[11].

La autenticación del cliente es más delicada, ya que exigir que se lleve a cabo mediante un certificado digital puede ser un requisito demasiado duro (desde el punto de vista de la usabilidad).

El mecanismo de autenticación más común en todos los servicios de almacenamiento en la nube es la utilización de contraseñas, aunque cada vez más servicios han ido incorporando mecanismos de autenticación multifactor, como Google Authenticator[12].

3.3 Deduplicación de datos

El concepto de deduplicación de datos se refiere a la gestión que hace el proveedor del servicio sobre los datos que se reciben dos o más veces, ya sea por parte del mismo usuario, o de usuarios distintos. En ese caso, el servidor puede o bien almacenar de nuevo dichos datos, o simplemente crear un enlace a los datos ya existentes.

Aunque este no sea un elemento que afecte directamente a los usuarios, es importante, ya que permite una gestión más eficiente del ancho de banda y de los sistemas almacenamiento (ya que podrán almacenar más ficheros o bloques utilizando menos bytes). Además, esto probablemente tenga una influencia directa en el precio que pagan los usuarios finales, ya que una gestión más eficiente de las comunicaciones y del almacenamiento implica menos gastos para el proveedor y esto, a su vez, es reflejado en un servicio más económico.

Este hecho no tiene importancia únicamente en términos económicos o de efectividad,

[11] Definición de ataque man-in-the-middle: https://www.owasp.org/index.php/Man-in-the-middle_attack
[12] http://es.wikipedia.org/wiki/Google_Authenticator

sino que también puede afectar a la privacidad del sistema. Para entender las razones, se explican a continuación las diferentes alternativas de deduplicación de datos.

3.3.1 Deduplicación en cliente vs deduplicación en servidor

Como se desprende del nombre, la deduplicación puede realizarse en el servidor, o directamente en el cliente. En la deduplicación en el lado del servidor, o «server side», el cliente envía los datos al servidor, y finalmente es este último quien ejecuta los algoritmos de deduplicación.

En la deduplicación en el lado del cliente, o «client side», normalmente el cliente aplica una función hash sobre el bloque o fichero, y envía el resultado al servidor. El servidor, al recibirlo, comprueba si ya tiene una copia de esos datos, de modo que únicamente se envían los datos en el caso de que el servidor no posea una copia del fichero. Ambos comportamientos se muestran de forma esquemática en la Figura 3 y la Figura 4.

Figura 3. Deduplicación en servidor.

Figura 4. Deduplicación en cliente.

De realizarse en el servidor, se incurre en un mayor coste de comunicaciones, y si los datos no han sido previamente cifrados por el usuario (con su clave personal), también se puede violar su privacidad. Por otra parte, de realizarse en el cliente, el usuario obtiene información sobre si el fichero ya existía en el sistema o no (en caso de deduplicación *cross-user*, explicada a continuación), información que un atacante podría utilizar para extraer información, como se observa en el estudio de Harnik et al [7].

3.3.2 Deduplicación *single-user* vs deduplicación *cross-user*

Con deduplicación *single-user*, únicamente se comprueban duplicidades entre los datos de un mismo usuario. Por el contrario, en deduplicación *cross-user*, esta comprobación se realiza entre todos los usuarios del sistema. En la Figura 5 se muestra un esquema de deduplicación *single-user* y en la Figura 6 el equivalente para deduplicación *cross-user*. Por simplicidad, únicamente se muestra el caso de deduplicación en servidor (para deduplicación en cliente la comprobación mostrada en las figuras seguiría el mismo

principio, pero en lugar de enviando el fichero, enviando su hash).

Figura 5. Deduplicación *single-user*.

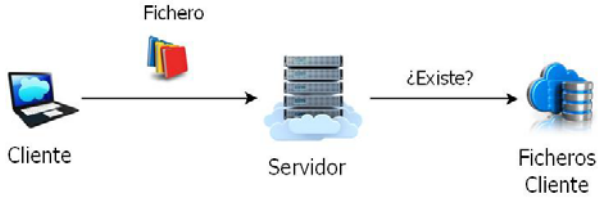
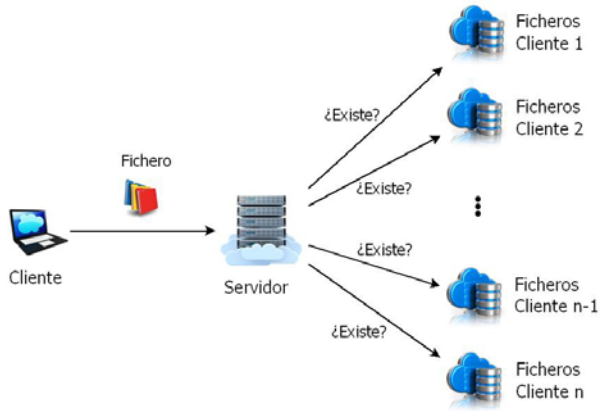


Figura 6. Deduplicación *cross-user*.



Con deduplicación *single-user* es más sencillo preservar la privacidad de la información de los usuarios, ya que no hay que realizar comprobaciones entre distintos usuarios y es más compatible con los algoritmos de cifrado. Por ejemplo, habitualmente el hecho de que distintos usuarios tengan distintas claves no afecta si los datos se cifran en el cliente, y se podría llevar a cabo indistintamente en el cliente o en el servidor. La contrapartida es que, como es lógico, los costes de comunicaciones y almacenamiento se incrementan.

La deduplicación *cross-user* es por lo tanto más eficiente en cuanto a costes, ya que permite un mayor ahorro en bytes transmitidos y almacenados. No obstante, este hecho es el origen de varias complicaciones.

- Por un lado, como se ha mencionado con anterioridad, cuando la deduplicación *cross-user* se realiza en el cliente, puede permitir a un atacante obtener información sobre qué datos almacenan otros usuarios.
- Por otro lado, suponiendo que (idealmente) cada usuario tenga su propia clave y que cualquier fichero es cifrado antes de ser enviado al servidor, si *N* usuarios suben el mismo fichero a la nube, se producirán *N* copias distintas para un

mismo fichero. Por lo tanto, la deduplicación del fichero ya no es posible, o al menos no siguiendo la solución tradicional.

La forma más directa de solucionar este problema, adoptada por algunos proveedores, es utilizar una misma clave común (controlada por el proveedor) para todos los clientes. Pero esto priva a los usuarios del control de sus datos, precisamente porque la clave utilizada no es conocida sólo por ellos.

Una técnica común que resuelve parcialmente este problema es la utilización de algoritmos de *cifrado convergente* (ver el *ANEXO 1 – Cifrado convergente* para un resumen de este tipo de cifrado, o la propuesta original en [8] para más detalles). Este tipo de cifrado produce, a partir de un mismo texto plano, el mismo texto cifrado, de forma que sólo puedan descifrar el texto cifrado quienes conozcan el texto plano original. Esto es compatible con que cada usuario disponga de una clave personal diferente, pero sigue permitiendo al servidor saber qué usuarios tienen los mismos ficheros, aunque no gane conocimiento sobre su contenido.

3.3.3 Deduplicación a nivel de fichero o a deduplicación a nivel de bloque

Esta variante indica a qué nivel de profundidad se lleva a cabo la deduplicación. En caso de realizarse a nivel de fichero, simplemente se des-duplican ficheros idénticos. Alternativamente, cuando la deduplicación es a nivel de bloque, los ficheros se dividen en bloques (normalmente de un tamaño predeterminado) y se realiza la misma comprobación por cada bloque que compone los ficheros.

En el caso de realizar deduplicación *cross-user*, y a nivel de fichero, cuando esta se produce, es directo deducir que algún otro usuario tiene exactamente el mismo fichero (si la deduplicación se calcula sobre los ficheros sin cifrar). Si la deduplicación se hace a nivel de bloque en lugar de a nivel de fichero, este hecho es posible utilizarlo como base para realizar un ataque que combine fuerza bruta con la elección de ficheros manipulados adecuadamente, con el fin de obtener ilegítimamente partes específicas de un fichero, como se indica en la Sección 2.2 del estudio de Harnik et al [7].

3.4 Compartición de datos entre usuarios

Algunos métodos, aunque no sean en sí mismos inseguros, pueden tener implicaciones en la seguridad y privacidad de los usuarios. Por ejemplo, si los ficheros se comparten a través de un enlace (funcionalidad ofrecida por muchos servicios), cualquier persona que tenga acceso a dicho enlace, o pueda predecirlo de alguna forma, puede acceder a la información asociada a menos que esta se encuentre cifrada. En otros casos, se puede compartir directamente el fichero a través de los contactos configurados dentro del propio servicio, es decir, sin generar ningún enlace público.

3.5 Borrado de datos

Un comportamiento típico en los proveedores de almacenamiento es mantener un fichero durante un número predeterminado de días después de que este haya sido borrado y antes de borrarlo definitivamente.

Otras políticas pueden afectar este comportamiento general. Por ejemplo, en el caso de que se produzca la deduplicación de ficheros entre usuarios (*cross-user*), se incrementa el contador de número de usuarios que poseen un fichero ya existente en el sistema, y se incluye una nueva referencia al mismo. De esta forma, aunque un usuario elimine «su» fichero, si otros usuarios también lo tienen en su cuenta, los datos reales no son borrados, sino simplemente la referencia al mismo.

Por último, al borrar un fichero en sí mismo (no una referencia), ¿qué tipo de borrado se hace?, ¿se lleva a cabo un borrado seguro (con múltiples sobre-escrituras) o simplemente se elimina el archivo o bloque? No obstante, probablemente no haya forma de determinar cuál de estas opciones aplica cada proveedor, y se deba confiar en que realicen una gestión adecuada del borrado en función de lo que declaran en sus condiciones de servicio.

3.6 Tipos de clientes

Idealmente, los distintos clientes implementados por un proveedor deberían ser equivalentes en cuanto a funcionalidad, aunque esto no es siempre así. Principalmente debido a factores como el ahorro en costes computacionales o de comunicaciones (importantes por ejemplo en móviles), o a las APIs disponibles normalmente limitadas desde los navegadores web, hay veces que se sacrifica la seguridad en pro de alcanzar mejores resultados en dichos aspectos, o en la usabilidad final de la aplicación. Por ejemplo, tal y como la aplicación *Bitcasa* especifica en la sección de seguridad de su página de información legal[13]:

> *For mobile clients, data may be encrypted on the server instead of the client.*
>
> (En clientes móviles, los datos pueden ser cifrados en el servidor en lugar de en el cliente.)

Por ello, especialmente si se va a hacer uso de distintos clientes, es importante tener en cuenta también este factor.

[13] https://www.bitcasa.com/legal.

3.7 Otros factores

Además de los aspectos ya mencionados, los proveedores de almacenamiento en la nube incorporan otros mecanismos para aumentar la eficiencia del sistema. Aun así, no se tiene constancia de que estos factores tengan influencia directa en la seguridad general del sistema (salvo que, como es evidente, haya vulnerabilidades en su implementación que puedan activar algún ataque). A continuación se enumeran dichos mecanismos:

1. Codificación basada en diferencias (generalmente llamada codificación delta o *delta encoding*). Consiste en enviar únicamente las diferencias de un fichero con la última versión almacenada, con el fin de alcanzar comunicaciones más eficientes.
2. Compresión. Como la palabra indica, consiste en comprimir la información antes de enviarla.

Por supuesto, también cabe mencionar que los diferentes proveedores pueden variar en las prestaciones puestas a disposición de los usuarios: capacidad de almacenamiento, costes de suscripción según el tipo de cuenta, funcionalidad específica (soporte para versionado, etc.). Aunque éstas son evidentemente importantes, no van a ser detalladas ya que no están directamente relacionadas con la seguridad.

Por otro lado, en el terreno legal, es importante conocer la normativa referente a los servicios de almacenamiento en la nube. De esta forma se podrá saber el respaldo que se tiene de las autoridades, o en qué casos éstas pueden verse legalmente autorizadas a acceder a la información que se sube.

En España, la ley que regula el tratamiento de datos personales en España es la LOPD [9]. No obstante, dado que la mayoría de los servicios en la nube no estarán localizados en España, también hay que conocer las normativas extranjeras. En el marco del Espacio Económico Europeo, la **directiva 95/46/EC**, del año 1995, sobre protección de datos, establece límites sobre el tratamiento de datos personales. En Estados Unidos, según [10], se deben tener en cuenta el USA PATRIOT ACT, originado tras los atentados terroristas del 11 de septiembre sobre la ciudad de Nueva York; y la doctrina de terceros de la **Cuarta Enmienda,** destinada a proteger la privacidad de los usuarios. Por último, también es relevante el Marco de Puerto Seguro (*Safe Harbor Framework*), un acuerdo de cooperación entre EE.UU. y la Unión Europea, para asegurar la transmisión segura de información entre ambos países.

4 Metodología de análisis

En la sección anterior se han introducido los aspectos importantes que pueden influir en la seguridad de una aplicación de almacenamiento en la nube. No obstante, a la hora de realizar un análisis, hay que centrarse en las acciones concretas que se pueden llevar a cabo en dichas aplicaciones. En este sentido, aunque algunos de los aspectos vistos son directamente acciones posibles dentro de una aplicación (como la compartición de ficheros y su borrado), otras son propiedades más transversales (como el cifrado y la autenticación). A continuación se dan unas pautas básicas sobre cómo analizar una aplicación de almacenamiento en la nube a partir de dichas propiedades.

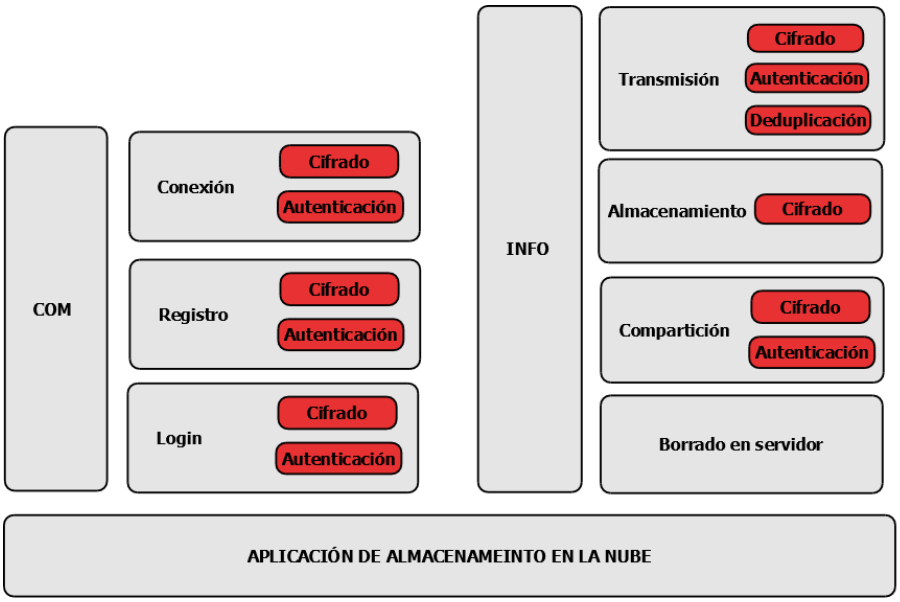
Las principales operaciones se pueden dividir en dos partes:

- Aquellas relacionadas con los protocolos de comunicaciones implementados. En concreto, es relevante el **tipo de conexión** que se utiliza, el proceso de **registro** para nuevos usuarios y cómo éstos hacen ***login*** posteriormente. Para estas operaciones, son especialmente importantes los mecanismos de cifrado y autenticación (ver secciones 3.1 y 3.2).
- Aquellas relacionadas con la gestión de la información (ficheros y carpetas), dado que es el principal uso de estas aplicaciones. Específicamente, es importante la manera en que se realiza la **transmisión de la información**, donde también serán esenciales los mecanismos de cifrado y autenticación (secciones 3.1 y 3.2) pero también, como se ha visto en la sección 3.3, la realización o no de deduplicación. Cómo se **almacena la información** una vez llega a los servidores de la compañía, en concreto, si se almacena cifrada o no, y de qué forma. Además, como se ha visto en las secciones 3.4 y 3.5, también hay que tener en cuenta cómo se realiza la **compartición de información** con otros usuarios y la **gestión del borrado** de la información subida.

También habrá que tener en cuenta los **tipos de clientes** (sección 3.6) que soporta la aplicación, realizando un análisis por separado para cada uno de ellos en los que pueda variar el comportamiento de alguna de las operaciones anteriores. Y, por supuesto, cada aplicación específica puede requerir tener en cuenta aspectos adicionales, como la forma de implementar funciones criptográficas (un aspecto conflictivo en Mega, como se verá en la sección 6.3.3).

A partir de esta organización, resumida en la Figura 7, se pueden analizar desde un marco común las distintas aplicaciones de almacenamiento en la nube.

Figura 7. Componentes para analizar en una aplicación de almacenamiento en la nube



En las secciones siguientes, se aplica esta metodología para analizar los casos específicos de los clientes web para Dropbox y Mega. No obstante, muchos de los aspectos tratados son comunes a todos sus clientes. En estos casos concretos, las herramientas que se han utilizado para el análisis, son:

- El proxy Squid[14], con la extensión SSL Bump[15], actuando como *Man In the Middle*.
- Wireshark[16] y la utilidad ZAP[17] de OWASP para analizar el tráfico.
- Firebug[18] como depurador web.

A partir del tráfico capturado y la información obtenida a través de depurar las aplicaciones, se ha extrapolado el funcionamiento de las mismas y la gestión de la información que transmiten.

[14] http://www.squid-cache.org/
[15] http://wiki.squid-cache.org/Features/SslBump
[16] https://www.wireshark.org/
[17] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[18] https://addons.mozilla.org/es/firefox/addon/firebug/

5 Análisis de Dropbox

Dropbox[19] ofrece cuentas gratuitas con capacidad de 2 GB (y hasta 16 adicionales a través de invitaciones) y cuentas Pro de 1 TB. Incluye soporte para Windows, Mac, Linux, dispositivos de Apple, Android y Blackberry. Utiliza *delta encoding* y permite compartir carpetas con otros usuarios, así como crear enlaces públicos para ficheros específicos.

Este servicio mantiene un historial de los cambios realizados durante el último mes, permitiendo recuperar versiones anteriores y hasta ficheros eliminados. En lo relativo a la seguridad, **protege las comunicaciones con el protocolo SSL y cifra los datos en los servidores con AES-256[20]**. En las cláusulas de seguridad, también se especifica que no todos los clientes móviles son compatibles con difusión cifrada de datos, por lo que **puede haber ocasiones en las que no se cifre determinada información antes de enviarlos** (se hace énfasis en los contenidos multimedia).

Por otro lado, a pesar de que Dropbox está adherido al Marco de Puerto Seguro[21] y a que hacen mucho énfasis en que sólo bajo condiciones excepcionales se permite el acceso a los datos por parte de sus empleados, Dropbox tiene acceso a la información almacenada por sus usuarios en su nube[22]. En concreto, indican que en caso de que la justicia lo requiera, podrán descifrar y remitirle la información almacenada.

También hay disponibles varias APIs [23] que permiten interactuar con distintos componentes del servicio y desarrollar aplicaciones propias.

5.1 Comunicaciones

5.1.1 Conexión

Tanto para conexiones web (ver Figura 8) como para conexiones con la aplicación de escritorio se utiliza cifrado AES de 128 bits, y como algoritmo de negociación de clave, Diffie-Hellman Efímero[24] autenticado con RSA. Esto último significa que:

- La clave simétrica se genera con el algoritmo Diffie-Hellman Efímero, es decir, utilizando nuevos valores públicos en cada negociación en lugar de valores fijos

[19] https://www.dropbox.com/es/.
[20] https://www.dropbox.com/security, apartado «Cifrado de datos en tránsito y en pausa».
[21] Ver https://www.dropbox.com/privacy y http://safeharbor.export.gov/companyinfo.aspx?id=17837.
[22] https://www.dropbox.com/help/27.
[23] https://www.dropbox.com/developers.
[24] http://wiki.openssl.org/index.php/Diffie_Hellman.

contenidos en un certificado digital. De esta forma, se consigue *Forward Secrecy*[25].

- El servidor utiliza su identidad digital, generada con RSA, para autenticar la negociación y evitar ataques de tipo *Man in the Middle*.

Figura 8. Tipo de conexión SSL con Dropbox.

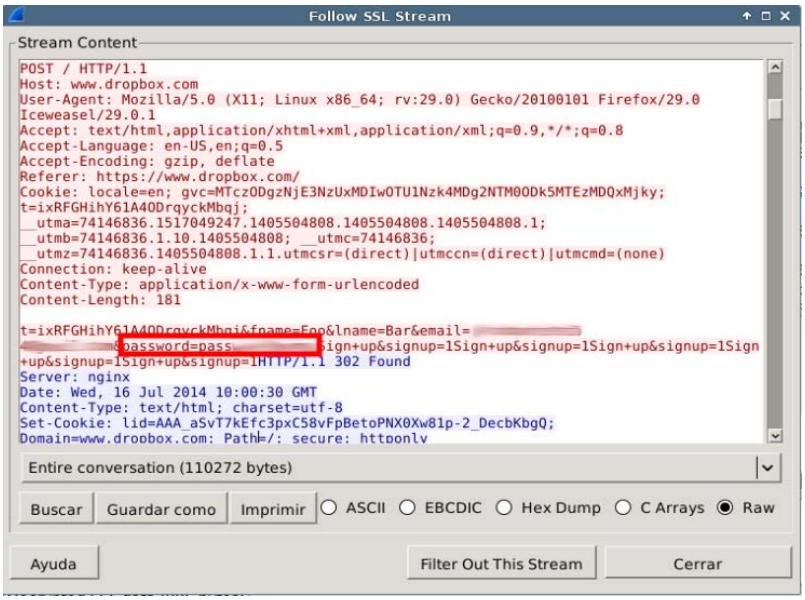
5.1.2 Registro

Para crear una cuenta en Dropbox desde su página web, es necesario introducir: nombre, apellido, dirección de email y contraseña, valores que son transmitidos al servidor (mediante una conexión TLS, como se ha mostrado con anterioridad). Es destacable que **la contraseña**, pese a enviarse a través de una conexión TLS, **es recibida por Dropbox sin procesar por una función hash o similar**, teniendo los servidores acceso a la contraseña en claro. Esto se puede observar en la Figura 9, en la que se muestra una captura del tráfico TLS (ya descifrado) de un proceso de registro, obtenida con Wireshark utilizando el proxy Squid [26] para poder descifrar la información intercambiada.

[25] http://en.wikipedia.org/wiki/Forward_secrecy.

[26] http://www.inteco.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/Analisis_trafico_SSL.

Figura 9. Captura de tráfico durante el registro en Dropbox.



Es decir, pese a que la comunicación está protegida con TLS, **Dropbox recibe la contraseña tal cual, en lugar de recibir un hash u alguna otra función de la misma**. Esto no es una buena práctica, ya que en caso de verse comprometidos los servidores de Dropbox por un atacante, éste podría directamente obtener las contraseñas de nuevos usuarios, independientemente de si estas son posteriormente cifradas para su almacenamiento. Hay varias alternativas para mejorar este proceso[27]. Por ejemplo, se podría limitar el impacto de compromisos del servidor utilizando hashes *salteados* de la contraseña. Alternativamente, este proceso sería notablemente más seguro si se utilizase el protocolo SRP[28] o similares.

5.1.3 Login

La autenticación básica es de tipo usuario y contraseña. En la aplicación web, igual que en el proceso de registro, la contraseña es enviada vía TLS, pero sin ser procesada previamente con una función hash o similar, teniendo los servidores acceso a la contraseña en claro. Es decir, **un atacante que comprometa los servidores de Dropbox podrá acceder a las contraseñas en claro de los usuarios que accedan a sus cuentas**.

Dropbox también permite configurar **autenticación multifactor**[29], lo cual añade mecanismos de autenticación adicionales además de la contraseña. Esta opción se

[27] https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/autenticacion_passwords.
[28] https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/autenticacion_passwords_srp
[29] https://www.dropbox.com/en/help/363.

puede gestionar desde las opciones de configuración de la página web, en la pestaña de seguridad. Existen dos alternativas al respecto:

- Indicar un número de teléfono móvil, al que se envía un código por SMS cada vez que sea necesario autenticar al usuario.
- Utilizar una aplicación de móvil[30], desde la que, la primera vez que se configure esta opción, se escanea un código QR para establecer las semillas que luego se utilizan para generar números aleatorios temporales. Cuando el usuario deba autenticarse, tiene que introducir el número aleatorio que muestre la aplicación.

Del mismo modo, como mecanismo para protección ante ataques CSRF[31], Dropbox establece un token de sesión distinto tras una autenticación satisfactoria, siguiendo el patrón de token sincronizador aconsejado por el OWASP[32] para evitar este tipo de ataques. Este token, de 18 bytes, se retransmite posteriormente en cada petición.

5.2 Gestión de la información

5.2.1 Transmisión

Como se ha indicado en la Sección 3.3, un aspecto importante que puede afectar a la privacidad de los usuarios es el hecho de si se realiza o no deduplicación entre usuarios (*cross-user*) al mandar información al servicio en la nube. En Dropbox, al transmitir ficheros, éstos se trocean por defecto en *fragmentos* de 8MB si se utiliza la interfaz web y en *fragmentos* de 4MB si se utiliza la API de Python, según la definición del método `upload_chunked` en el fichero *client.py* de la misma:

```
def upload_chunked(self, chunk_size = 4 * 1024 * 1024):
```

Para comprobar si se produce en este caso, se ha procedido siguiendo la metodología descrita en la investigación de Harnik et al [7]. Es decir, se ha comprobado la cantidad de datos transmitida y el tiempo total de transmisión para distintos ficheros, desde la misma y desde diferentes cuentas de usuario. En concreto, las pruebas realizadas se han llevado a cabo de la siguiente manera:

1. Se ha creado un fichero aleatorio (`rnd.100` en el ejemplo siguiente) de gran tamaño para minimizar la probabilidad de que ya exista en la cuenta de algún

[30] Por ejemplo, Google Authenticator.
[31] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).
[32] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#General_Recommendation:_Synchronizer_Token_Pattern.

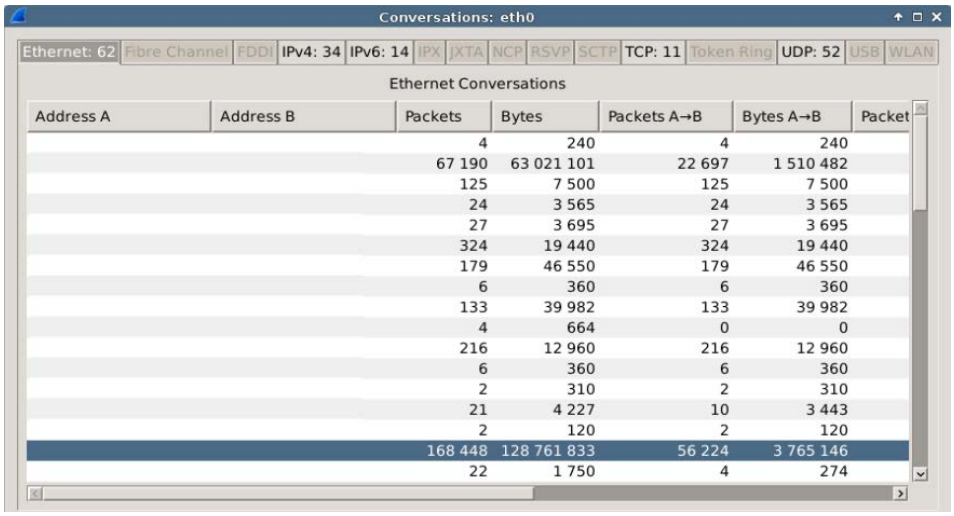
usuario de Dropbox. Por ejemplo, desde Linux se pueden generar ficheros aleatorios de la siguiente forma (en este caso, de 100 MB):

```
$ gpg --gen-random 0 102400000 > rnd.100
```

Nótese que, aunque se especifica que la calidad de los números aleatorios generados sea baja (el parámetro `0`), en este caso no es importante, ya que no se pretende dar una funcionalidad criptográfica a dichos números.

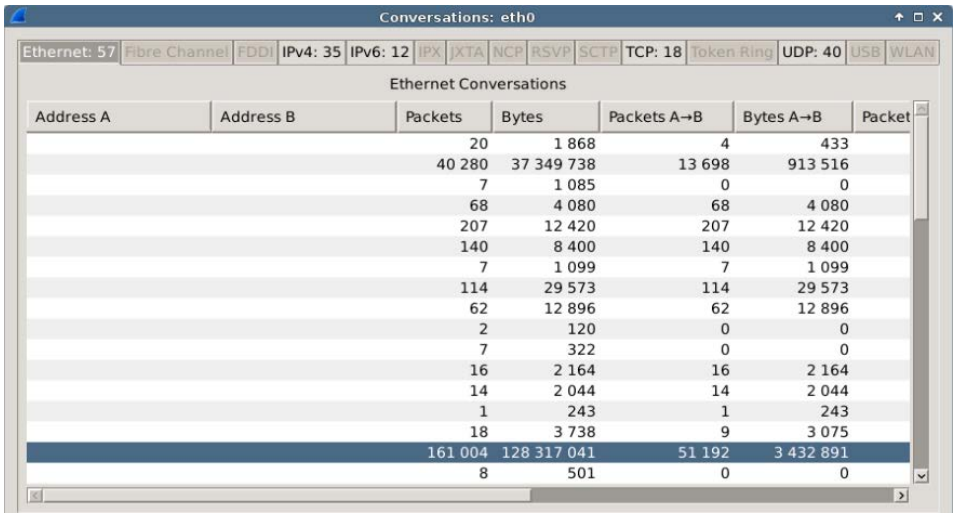
2. A continuación, se ha copiado el archivo `rnd.100` en la carpeta de Dropbox de una cuenta *A*, midiendo la cantidad de bytes transmitidos al sincronizar con los servidores de Dropbox. Como se muestra en la Figura 10, de entre todas las conversaciones capturadas por Wireshark, la conversación resaltada corresponde a la sesión de envío del fichero. En ella se han transmitido aproximadamente 122MB.

Figura 10. Número de bytes transmitidos al subir el fichero `rnd.100` a la cuenta *A*.



3. El mismo archivo `rnd.100` se ha copiado en la carpeta de Dropbox de una cuenta *B*, midiendo de nuevo la cantidad de bytes transmitidos. De nuevo, en la Figura 11, se observa que la cantidad de bytes transmitidos es prácticamente la misma. Esto es indicativo de que **no se realiza deduplicación *cross-user* en el cliente** (ni a nivel de bloque, ni a nivel de fichero).

Figura 11. Cantidad de bytes transmitidos al subir el fichero rnd.100 a la cuenta B.



Siguiendo el mismo procedimiento (con un fichero `rnd.100.2` distinto), pero para una única cuenta de usuario, se comprueba que Dropbox sí hace deduplicación *single-user*, tal y como se muestra en la Figura 12 y la Figura 13. Se observa que la primera vez que se sube el fichero (Figura 12) se envían aproximadamente 110 MB, mientras que en el segundo envío (Figura 13), se mandan únicamente alrededor de 0,02 MB.

Figura 12. Envío del fichero rnd.100.2 a la cuenta A.

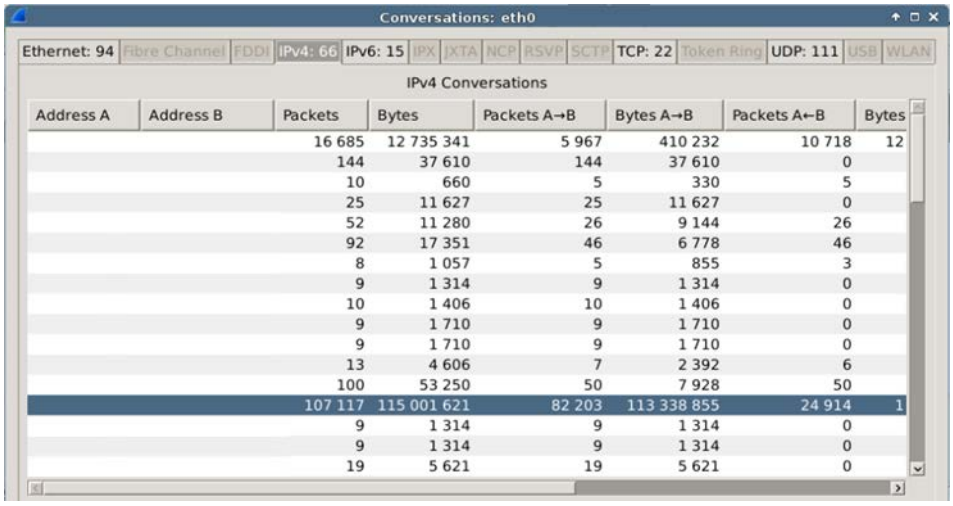
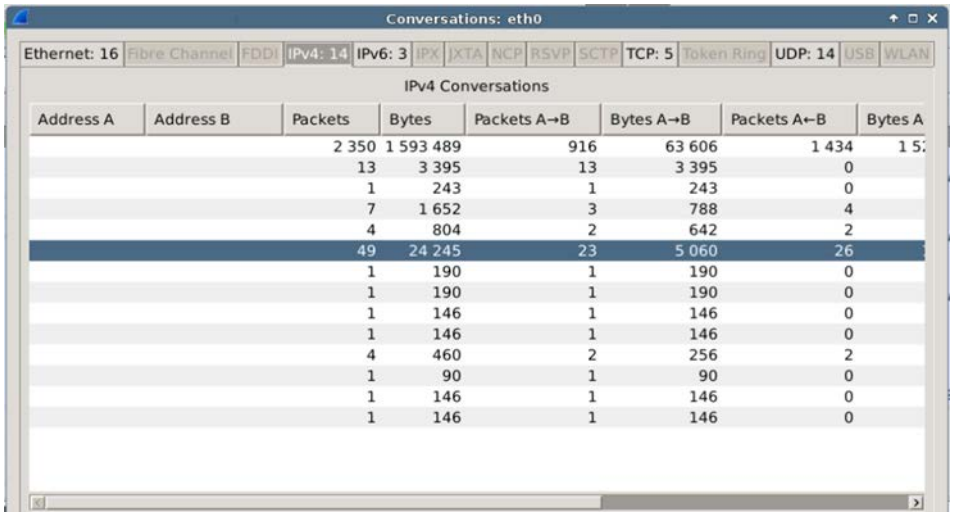


Figura 13. Re-envío del fichero rnd.100.2 a la cuenta A.



5.2.2 Almacenamiento

Según la compañía[33], la información subida a su nube se almacena cifrada con AES de 256 bits. También afirman[34] que únicamente un pequeño número de empleados tiene acceso a las claves utilizadas para cifrar la información.

5.2.3 Compartición

Tanto desde el cliente de escritorio como el cliente web, se puede compartir ficheros o directorios generando un enlace a los mismos. Los enlaces generados tendrán la forma:

https://www.dropbox.com/s/<valor aleatorio>/<nombre fichero>?dl=0

Normalmente, cualquiera que disponga del enlace podrá acceder al fichero o directorio, aunque en cuentas de pago sí es posible establecer restricciones de visibilidad, estableciendo una contraseña o configurando una fecha de caducidad del enlace.

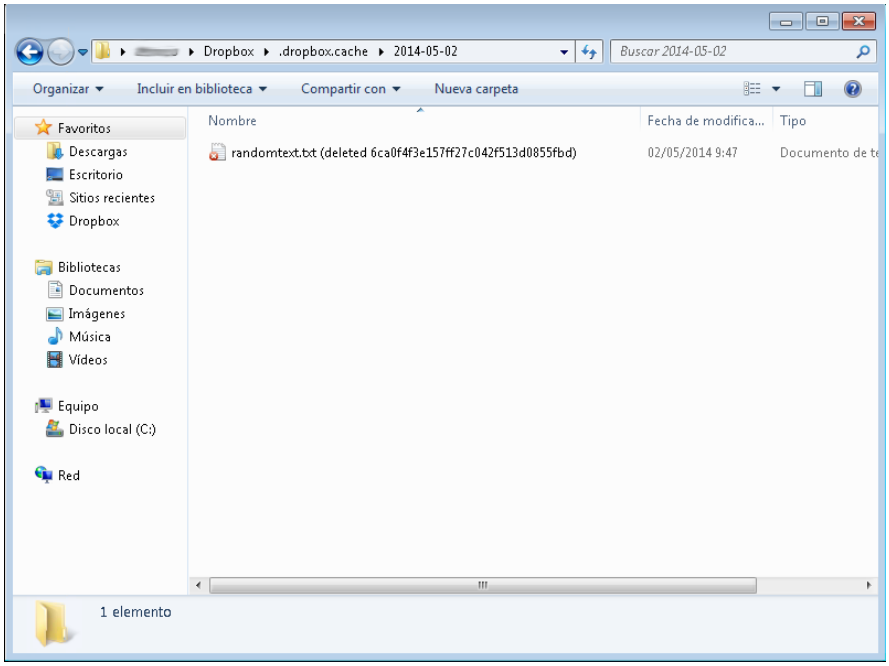
5.2.4 Borrado

Según declara Dropbox[35], en cuentas estándar, los ficheros borrados se mantienen durante 30 días, pudiendo ser recuperados en ese intervalo desde la interfaz web. Para cuentas que incluyan la opción de «*Extended Version History feature*», este periodo se incrementa a un año.

[33] https://www.dropbox.com/security.
[34] https://www.dropbox.com/help/27.
[35] https://www.dropbox.com/help/296.

En caso de utilizar la aplicación de escritorio, Dropbox mantiene un directorio oculto, *.dropbox.cache*, donde se almacenan los ficheros borrados, en subcarpetas ordenadas por fecha. En este caso, si se tiene la cuenta configurada en varios dispositivos, al borrar un fichero en uno de ellos, el resto se actualiza consecuentemente. Por ejemplo, si el fichero *randomtext.txt* ha sido borrado el 2 de mayo de 2014, la carpeta creada tendrá el nombre 2014-05-02, con un contenido similar al mostrado en la Figura 14.

Figura 14. Ficheros borrados en el cliente de escritorio de Dropbox.



5.3 Otros aspectos

Más allá de las propiedades comentadas hasta ahora, relativas a las forma en que Dropbox protege la información de sus usuarios, *at rest* y en tránsito, también hay otras medidas de seguridad destacables. Por ejemplo:

- En dispositivos móviles, Dropbox permibrote establecer códigos de desbloqueo propios[36], lo cual añade una capa de seguridad al código o patrón de desbloqueo que pueda estar activo en el dispositivo.
- Desde la interfaz web, es posible gestionar todos los dispositivos vinculados[37]. Así, si por ejemplo se pierde un «smartphone» en el que estuviera instalado Dropbox, puede desvincularse la cuenta de forma remota para revocar el

[36] https://www.dropbox.com/help/227.
[37] https://www.dropbox.com/help/25.

acceso a través del dispositivo extraviado y proteger así la confidencialidad de los ficheros.

5.3.1 Gestión personal de claves

Poder crear claves propias desde programas externos y posteriormente importarlas en Dropbox supondría una mejora notable para usuarios que ya disponen de sus propias claves. Además, daría garantías adicionales para proteger los datos del usuario *at rest* ya que éstos estarían cifrados con la clave del propio usuario, en lugar de con una clave fuera del control del mismo. En la actualidad, **Dropbox no soporta esta funcionalidad**, aunque no descartan añadirla en un futuro[38].

5.4 Resumen

En base a lo comprobado en este estudio, las principales características que pueden influir en la seguridad del servicio proporcionado por Dropbox, son:

- **Conexión**: TLS 1.2, con AES de 128 bits en modo GCM, ECDHE y RSA para negociación de claves.
- **Registro**: La contraseña es enviada a través de TLS, pero sin procesar por una función hash o similar, teniendo los servidores acceso a la contraseña en claro.
- **Autenticación y *login***:
  - La contraseña es enviada a través de TLS, pero sin procesar por una función hash o similar, teniendo los servidores acceso a la contraseña en claro.
  - Soporte para autenticación de doble factor.
- **Transmisión de ficheros**:
  - Realiza deduplicación *single-user* en cliente, a nivel de fichero.
  - No realiza deduplicación *cross-user* en cliente.
  - No se ha podido comprobar si lo hace en el servidor.
- **Almacenamiento de ficheros**: Cifrados con AES de 256 bits. Claves controladas por Dropbox.
- **Compartición de ficheros**:
  - A través de enlaces. Para clientes gratuitos, no se pueden proteger con contraseña o de alguna otra forma.
  - A través de contactos.
- **Borrado de ficheros**: Según sus términos de servicio y para clientes gratuitos, los ficheros se borran del servidor a los 30 días de haberlos borrado el usuario.

[38] https://www.dropbox.com/help/28/en (accedido por última vez el 09/09/2014).

- **Otros aspectos**:
  - **Móviles**: Posibilidad de añadir código de bloqueo propio.
  - **Compatibilidad con gestión propia de claves**: No.

6 Análisis de Mega

Mega[39] proporciona un cliente web, afirmando que funciona con cualquier navegador moderno, aunque aconsejan utilizar Google Chrome, Mozilla Firefox u Opera Next[40]. También hay clientes móviles para Android, Blackberry e iOS y clientes Sync (para sincronización de escritorio) compatibles con Linux, Mac y Windows. Adicionalmente, proporciona una API, lo que permite la creación de clientes no oficiales. Para cuentas gratuitas, el límite de almacenamiento es de 50 GB, mientras que las cuentas de pago varían desde 500 GB hasta 4 TB de almacenamiento, junto con anchos de banda disponibles crecientes.

Uno de los principales reclamos de Mega es haber contado con la privacidad de sus usuarios desde su diseño («privacy by design»). En concreto, lo más destacable es que la funcionalidad criptográfica destinada a proteger la información almacenada en los servidores de Mega es ejecutada en el ordenador cliente, a través de código JavaScript descargado de sus propios servidores. De esta forma, en Mega afirman que salvo que un usuario haga público el enlace a un fichero almacenado en sus servidores, junto con la clave utilizada para cifrarlo, ellos no tendrán forma de acceder al contenido.

Nota: En algunos de los apartados siguientes, se hace referencia a ficheros con el código fuente de Mega. El nombre de estos ficheros suele seguir el formato `<nombre>_<número>.js`, donde `<número>` es un número entero que va aumentando conforme Mega introduce cambios en el fichero. Por lo tanto, algunos de estos números probablemente cambien desde la publicación de este documento.

6.1 Comunicaciones

6.1.1 Conexión

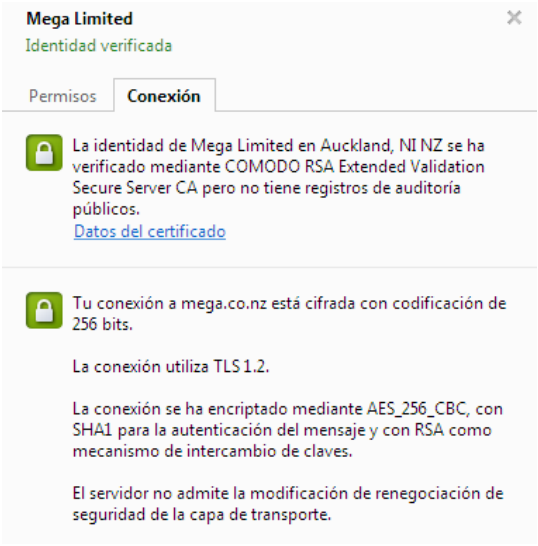
En las conexiones web Mega utiliza la versión 1.2 de TLS, con un cifrado AES de 256 bits, en modo CBC, y como algoritmo de negociación de clave, utiliza RSA.

Al utilizar RSA para negociar la clave simétrica implica que ésta es generada por el cliente, cifrada con la clave pública contenida en el certificado presentado por el servidor (emitido por Comodo, como se puede ver en la Figura 15), de tipo RSA y de

[39] https://mega.co.nz.
[40] https://mega.co.nz/#help.

2048 bits. De esta forma, sólo Mega puede descifrar la clave TLS generada por el cliente y se evitan ataques de tipo *Man in the Middle*.

Figura 15. Tipo de conexión TLS con Mega.



6.1.2 Registro

Los datos que deben proporcionarse para realizar el registro de una cuenta en el servicio son: nombre, apellido, dirección de email y contraseña, además de aceptar las condiciones del servicio. Al enviar el formulario con estos datos, entre otras acciones, se crea un usuario anónimo (a falta de confirmar el email indicado), para el cual se genera una clave maestra de usuario de 4 enteros (32 bytes).

La clave maestra se cifra con AES utilizando una clave derivada de la contraseña introducida por el usuario, y se envía de este modo. El código JavaScript que genera estos valores, y un ejemplo de petición, se pueden ver en la Figura 16.

Figura 16. Fragmento de la función *api_createuser* en Mega y ejemplo de valores generados.

```
if (!u_k) api_create_u_k();

for (i = 4; i--; ) ssc[i] = rand(0x100000000);

if (d) console.log("api_createuser - masterkey: " + u_k + " passwordkey: " + ctx.passwordkey);

req = { a : 'up',
        k : a32_to_base64(encrypt_key(new sjcl.cipher.aes(ctx.passwordkey),u_k)),
        ts : base64urlencode(a32_to_str(ssc) + a32_to_str(encrypt_key(new sjcl.cipher.aes(u_k),ssc))) };
```

[{"a":"up","k":"OR8Kx0rhnPH8frI3mj3HhQ","ts":"EQniVKoDgiSibnDOrX_-QzSmpAgUjdnt8oDlkGHPcx4"}]

La petición de registro es respondida con un email de activación, enviado a la dirección especificada. En la Figura 17 se muestra la función encargada de este envío, junto con un ejemplo de la respuesta enviada al ordenador cliente y el email asociado que es recibido por el usuario.

Figura 17. Envío de email de confirmación en Mega. Función `sendsignuplink` y ejemplo de datos enviados al cliente a través del navegador y por email.

```
function sendsignuplink(name,email,password,ctx)
{
    var pw_aes = new sjcl.cipher.aes(prepare_key_pw(password));
    var req = { a : 'uc',
                c : base64urlencode(a32_to_str(encrypt_key(pw_aes,u_k))+a32_to_str(encrypt_key(pw_aes,[rand(0x100000000),0,0,rand(0x100000000)]))),
                n : base64urlencode(to8(name)),
                m : base64urlencode(email) };

    api_req(req,ctx);
}
```

[{"a":"uc","c":"PPAhJpTNowHBemxsMHdfWrVGxJ8mAU8uAOwu3YIChM0","n":"Rm9vIEJhcg","m":"Zm9vYmFydGVzdHMyQGhvdG1haWwuY29t"}]

Welcome to MEGA! Please click on the following link to confirm your free MEGA account:

https://mega.co.nz/#confirmPPAhJpTNowHBemxsMHdfWrVGxJ8mAU8uAOwu3YIChM2x_krNnQIADGa3dIYq9U5mb29iYXJ0ZXN0czJAaG90bWFpbC5jb20JRm9vIEJhcrxghq4cTG_p

Al acceder al enlace recibido por email, se pide al usuario que vuelva a introducir la contraseña indicada al solicitar el registro. En caso de introducir la contraseña correcta, se envía al servidor la dirección de correo electrónico cifrada con una clave AES derivada de la contraseña, tal y como se muestra en la Figura 18. Este valor («uh» en la figura), se utiliza posteriormente para autenticar al usuario a modo de *token* de autenticación.

Figura 18. Envío de la dirección de email registrada, cifrada con la clave AES derivada de la contraseña

```
{
  uh : stringhash(email.toLowerCase(),passwordaes),
  c: confirmcode
})
```

Por último, se genera un par de claves RSA (de 2048 bits), enviándose finalmente la clave privada cifrada con la clave AES maestra del usuario, tal y como se muestra en la Figura 19.

Figura 19. Envío del par de claves RSA, con la clave privada cifrada con AES y ejemplo de valores generados.

```
api_req({
            a : 'up',
            privk : a32_to_base64(encrypt_key(u_k_aes,str_to_a32(crypto_encodeprivkey(rsakey)))),
            pubk : base64urlencode(crypto_encodepubkey(rsakey))
        },
        ctx);
```


```
[{"a":"up","privk":"4TG22JrhuxzErFoTgKsbkBRLoabAXuvES93ClYLMBKlENwWk2ykRUWr1kkBq6kV0T74BM2aYCca1IdY71TGxw3ZCjHZfV8RM8xYjYCgT3PBsTmrGzundq2FbGSP43L
2yV2DjmAiMyQdsizySddeuU7lu6W2xJpspbmUHjc4ZfAJoiwvp6xjU40lCH7lt3KDCW6jRNtrGH_-F6JFImgBuebcC85hgLBeGiydLh9vuOEst2vsfGFr70qEAM1hDzuNmRc2Axfzrv_IS5cB-
msj7segi_3RKUWgMMRlNfQX22DsV3S5CsNfndzu9eeIm9HpXG3y1LgUbSdrwpcCUL--oNaHZqPDQKfGbTu6-rKGsL39sXA0tQIK7xi1KuZMwiQdeghY2t-IQtInexP2JCVvQ_
zgVZW3JGJtXbpWaa9YpvqYHeMLHmEqCciqqJnSY1GyDAq0xoK98_bDxWVux7Z4ut3DVNl73yZPVPGwooTq9HCUP8YvyBEMBgdf0STWSXIh4jfrI3E3up4TJ0yWlgJvVyGzWxV6EeZhgADzTDnv
AcO3QlxP9hpHFTNm4T10XlsjJ5XmaDrrpZ5lRHHS_PTvSnKUHsSLt1ETsV_kyTOSrv2ULTHyAwOm8PD7N-tONjyT7t8azRN3swSZqF2fDPjgVKpqUVun5UO9A2HgUycs7LpTbC6kS9S9As6ULB
JnN49c7rj8MAyrTI249TlluCd9Qt8pcX_nppSi1tYSycf4F1SJ1B3bv-Rb6LSqlEgNWUXuHK3jw1hUPiFMaY","pubk":"CACAEilgzZMOo0tuE5EIWswQCf4YU4viMpQVChs6fDeLZ4w1H6E5
4Zf1ZRvNYAFdlqCq4fUDtLOL16tadmtX148kr7RXZCJr9t2e2ydgdXekxI9t3zTY6k610rkAxpfrMrgcEnVIB-LihcKj6SLdyXvDuXtq_8LnzAUA5sifIaZR1s_AZqwC_nAW6RzjPbVsiOiOfJ
sovxVM1mIrbvEIf1F1w1bf4hmKrEHSa_519GvdeWueexFmlYN9kxhv9ePa-BQI2wWu2T2vhPrktXwWvk8k88FIbKBAWod_6w4BKsRQC2BiCM27mWIpcs3kSeXACAAAAEB"},{"a":"uga","u"
```


En resumen, durante el proceso de registro, se generan dos claves:

1. Una clave AES, de 256 bits, referida durante este estudio como clave AES maestra.
2. Un par de claves RSA, de 2048 bits.

La clave privada RSA se cifra con la clave AES maestra que, a su vez, es cifrada con una clave AES de 256 bits derivada de la contraseña introducida por el usuario. En esta forma cifrada, tanto el par RSA como la clave AES maestra se suben al servidor.

6.1.3 Login

Al hacer *login*, como se muestra en la Figura 20, desde el cliente se vuelve a cifrar la dirección de email indicada por el usuario con la contraseña introducida, del mismo modo que se hace durante el registro (ver Figura 18). En caso de introducir la dirección y contraseña correctas, el valor «uh» generado es correcto por lo que el servicio busca los datos del usuario y devuelve las claves cifradas del usuario.

Figura 20. Cálculo del token de autenticación al hacer login.

```
function postLogin(email,password,remember,callback)
{
    var ctx =
    {
        callback2:callback,
        checkloginresult: function(ctx,r)
        {
            if (ctx.callback2) ctx.callback2(r);
        }
    }
    var passwordaes = new sjcl.cipher.aes(prepare_key_pw(password));
    var uh = stringhash(email.toLowerCase(),passwordaes);
    u_login(ctx,email,password,uh,remember);
}
```

A continuación, el navegador del usuario:

1. Descifra la clave AES maestra utilizando la contraseña introducida por el usuario.
2. Descifra la clave privada RSA utilizando la clave AES maestra.

En la Figura 21 se muestra el fragmento de código en el que se realizan estas operaciones.

Figura 21. Fragmento de la función *api_getsid2* donde se descifran las claves de usuario durante el *login*.

```
var aes = new sjcl.cipher.aes(ctx.passwordkey);

// decrypt master key
if (typeof res.k == 'string')
{
    k = base64_to_a32(res.k);

    if (k.length == 4)
    {
        k = decrypt_key(aes,k);

        aes = new sjcl.cipher.aes(k);

        if (typeof res.tsid == 'string')
        {
            t = base64urldecode(res.tsid);
            if (a32_to_str(encrypt_key(aes,str_to_a32(t.substr(0,16)))) == t.substr(-16)) r = [k,res.tsid];
        }
        else if (typeof res.csid == 'string')
        {
            var t = base64urldecode(res.csid);

            var privk = crypto_decodeprivkey( a32_to_str(decrypt_key(aes,base64_to_a32(res.privk))) );
```

Finalmente, es importante reseñar que la plataforma no soporta autenticación de doble factor.

6.2 Gestión de la información

6.2.1 Transmisión

Como se puede observar en la Figura 22, Mega utiliza AES de 256 bits en modo CCM para la transmisión de ficheros. CCM es un modo de cifrado en bloque definido en el RFC 3610[41], que combina el modo CTR (Counter) con el modo CBC-MAC, proporcionando autenticidad además de confidencialidad.

Para cada fichero que se sube al servicio, se genera una nueva clave AES aleatoria. Al subir el fichero, esta clave se cifra con la clave AES maestra del usuario y se sube también a Mega, tal y como se muestra en la Figura 23, donde la clave que se utiliza para cifrar el fichero se envía en el campo «k». De este modo, el usuario puede recuperar el fichero desde cualquier sitio (recuérdese que la clave AES maestra del usuario se almacena también en servidor, cifrada utilizando la contraseña personal).

Figura 22. Fragmento del fichero *encrypter.js* para cifrar los ficheros subidos a Mega.

```
for ( var i = 0; i < data.length; i += 0x100000 )
{
    // put data chunk into the heap
    var j = ( i + 0x100000 < data.length ) ? i + 0x100000 : data.length;
    heap.set( data.subarray(i,j), 0x1000 );

    // init mac state
    asm.init_state.apply( asm, iv );

    // decrypt data
    asm.ccm_encrypt(0x1000, j-i,nonce[0], nonce[1], nonce[2], nonce[3], nonce[4],
                    nonce[5], nonce[6], nonce[7],0, 0, 0, 0, 0, 0,(ctr/0x100000000) >>> 0,ctr >>> 0);

    // get decrypted data from the heap
    data.set( heap.subarray( 0x1000, 0x1000+j-i ), i );

    // store mac
    asm.save_state(0x1000);
    macs.push( heapView.getUint32( 0x1000, false ) );
    macs.push( heapView.getUint32( 0x1004, false ) );
    macs.push( heapView.getUint32( 0x1008, false ) );
    macs.push( heapView.getUint32( 0x100c, false ) );
```

[41] http://tools.ietf.org/html/rfc3610.

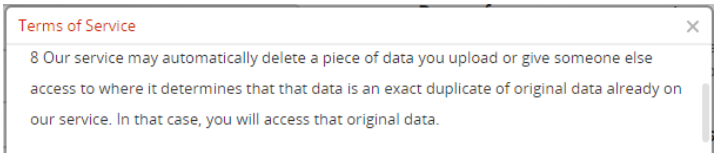
Figura 23. Fragmento de la función *ul_finalize* para el envío de clave de cifrado de fichero y ejemplo de envío.

```
var req = { a : 'p',
    t : dir,
    n : [{
        h : file.response,
        t : 0,
        a : ab_to_base64(ea[0]),
        k : a32_to_base64(encrypt_key(u_k_aes, file.filekey))
    }],
    i : requesti
};
```

```
[{"a":"p","t":"zwckUQDa","n":[{"h":"0YF8zZtu1nx5HHrydaWNg_2Is_AgydTYZr-7","t":0,"a":"aQ9ndeop1R0jMo9JSPA-bD9Jh51lPK-XlPV_-Ukj3Uxbw3xjBL5MNOVHr3M_L
wuGHTzJ3-Wh1VH1YYrSK-0NRrN-kwXRSq1TjyLmzmw8"}],"i":"KVkX8aZvSb"}]
```

Sobre la **deduplicación**, en su declaración de privacidad y seguridad (ver el fragmento correspondiente en la Figura 24), Mega se reserva el derecho de borrar un archivo si comprueba que es un duplicado exacto de otro archivo que ya esté en alguna otra ubicación de sus servidores.

Figura 24. Fragmento de las condiciones de servicio de Mega relativo a la deduplicación.



Pese a que este aspecto no aclara la política de deduplicación, sí es indicativo de que ésta probablemente se produzca en algún momento. Según el propio blog de Mega[42], se produce deduplicación una vez se han cifrado los datos. No obstante, dado que dicha entrada del blog es relativamente antigua (enero de 2013), y no parece haber más información específica y contrastada en otros recursos accesibles, se han realizado varias comprobaciones, para confirmar cómo realiza Mega la gestión de ficheros duplicados.

Por un lado, en la función encargada de subir los ficheros, localizada en el fichero `upload2_21.js`, se ejecuta el fragmento de código mostrado en la Figura 25. Dentro de la función *fingerprint* (localizada en el fichero `crypto_49.js`) que se invoca durante este proceso, se calcula el CRC del fichero (sin cifrar) a subir, al que concatena la fecha de última modificación (*lastModifiedDate*) del fichero, codificando el resultado en Base64, tal y como se muestra en la Figura 26. A continuación, la función *Finish* invoca

[42] https://mega.co.nz/#blog_3.

la función *callback*, que corresponde con la función anónima pasada como segundo parámetro a *fingerprint*. En esta función de callback se comprueba si, entre los ficheros del propio usuario, ya existe un fichero con el mismo «hash» (que en este caso es el CRC concatenado con la fecha de modificación). En caso de que ya existiese, se ejecuta la función *ul_deduplicate*, enviando los datos mostrados en la Figura 27, donde «*n.h*» es el hash calculado.

Por todo ello, se puede deducir que **Mega realiza deduplicación *single-user* en cliente a nivel de fichero**. Es decir, en caso de detectar un fichero duplicado dentro de la propia cuenta del usuario, evita el envío del fichero, aplicando el mecanismo indicado. En cualquier caso, cabe destacar que, pese a realizarse la deduplicación en cliente sobre los datos sin cifrar, al tratarse de deduplicación *single-user*, a priori esto no supone un riesgo para la privacidad del usuario.

Figura 25. Fragmento de código para deduplicación *single-user*.

```
try {
    fingerprint(file, function(hash, ts) {
        file.hash = hash;
        file.ts   = ts;
        var identical = ul_Identical(file.target, file.path || file.name, file.hash, file.size);
        DEBUG(file.name, "fingerprint", M.h[hash] || identical)
        if (M.h[hash] || identical) ul_deduplicate(self, identical);
        else ul_start(self);
    });
} catch (e) {
```

Figura 26. Parte final del cálculo de la *fingerprint* de los ficheros a subir.

```
function Finish(crc)
{
    callback(base64urlencode(crc+serialize((uq_entry.lastModifiedDate||0)/1000)),((uq_entry.lastModifiedDate||0)/1000));
}
```

Figura 27. Datos enviados en caso de duplicidad *single-user* y ejemplo de valores generados.

```
api_req({a:'g',g:1,ssl:use_ssl,n:n.h},

[{"a":"g","g":1,"ssl":2,"n":"3l8TEB4A"}]
```

Después de realizar esta comprobación, si el servicio no detecta duplicidades dentro de los ficheros del propio usuario, procede al envío del mismo. Por otro lado, en el presente estudio se ha comprobado que, como declara en su blog**, Mega también realiza**

deduplicación *cross-user* en servidor sobre los ficheros una vez cifrados.

Tras enviar el propio fichero, se envía al servidor meta-información sobre el mismo. En concreto, se envían los datos mostrados en la Figura 28, correspondiente a un fragmento de la función *ul_finalize*, localizada en el fichero `upload2_24.js`. Los datos importantes en este aspecto son los asociados al campo «`a`» y al campo «`k`», siendo este último la clave AES utilizada para cifrar el fichero, cifrada a su vez con la clave maestra del usuario. El primero es el nombre del fichero y su hash, ambos cifrados con la clave utilizada para cifrar el fichero. Este hash es el mismo que el utilizado en la deduplicación *single-user*, es decir, es un CRC32 calculado, entre otros elementos, a partir de los datos del fichero en plano. En este caso, nótese que la función `enc_attr`, cifra estos datos con la clave del fichero antes de enviarlos.

Figura 28. Fragmento de la función `ul_finalize`, con la meta-información sobre el fichero subido impllicada en la deduplicación *cross-user*.

```
var body  = { n: file.name }
if (file.hash) body.c = file.hash
var ea  = enc_attr(body
var faid = file.faid ? api_getfa(file.faid) : false
var req = { a : 'p',
    t : dir,
    n : [{
        h : file.response,
        t : 0,
        a : ab_to_base64(ea[0]),
        k : a32_to_base64(encrypt_key(u_k_aes, file.filekey))
    }],
    i : requesti
};
```

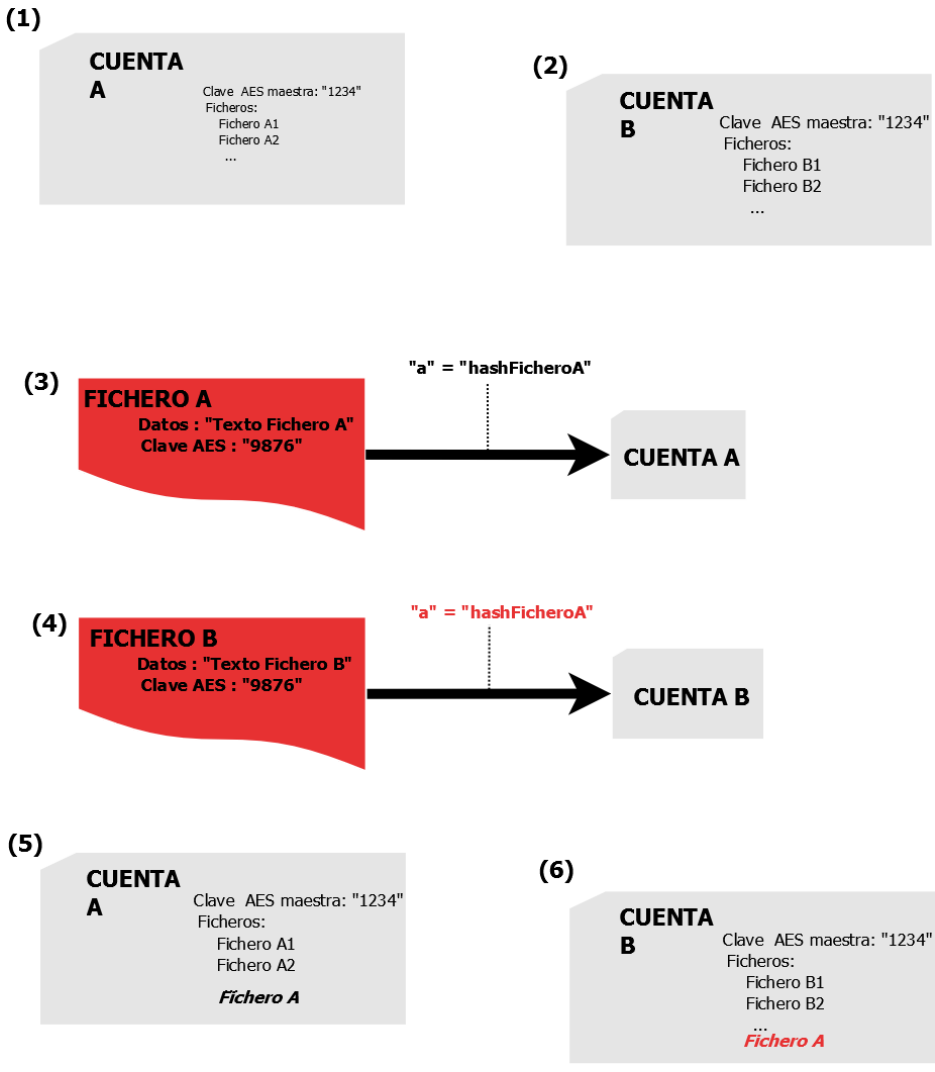
Se puede comprobar que se realiza deduplicación *cross-user* en servidor manipulando los campos «`a`» y «`k`» mencionados. Para ello, es necesario disponer de dos cuentas «preparadas» para tal fin (Cuenta A y Cuenta B, a continuación). Recuérdese que en Mega hay, además de la contraseña personal de cada usuario, tres tipos de claves: la clave AES maestra de usuario; claves AES para el cifrado de ficheros, que son a su vez cifradas con la clave maestra de usuario; y claves RSA para compartición de ficheros (esto se ve más adelante).

- En primer lugar, tanto la Cuenta A como la Cuenta B deben tener la misma clave AES maestra: esto se puede conseguir, por ejemplo, depurando la aplicación web durante el registro a la hora de crear la segunda cuenta y modificando los valores apropiados.
- En segundo lugar, son necesarios dos ficheros distintos: un Fichero A para subir a la Cuenta A, y un Fichero B para subir a la Cuenta B. A continuación, se asume

que primero se sube el Fichero A a la Cuenta A, y que posteriormente se sube el Fichero B a la Cuenta B.

- En tercer lugar, es necesario que para ambos ficheros se genere la misma clave AES. De nuevo, esto último se puede conseguir depurando la aplicación web durante la subida del Fichero B, y modificando los valores apropiados.
- Finalmente, al subir el fichero, es necesario modificar el valor enviado en el campo «a» del Fichero B, estableciéndolo al mismo valor que se envió para el Fichero A. Una vez realizado esto, el fichero que se descargue desde la Cuenta B será el Fichero A, en lugar del Fichero B. Este proceso se esquematiza en la Figura 29.

Figura 29. Esquema de la prueba de concepto para comprobar la deduplicación *cross-user* en Mega.



Sobre este aspecto, es necesario hacer varias puntualizaciones:

- El requisito de hacer coincidir las claves es únicamente para que la Cuenta B muestre correctamente el Fichero A. En caso de no cambiarse, la deduplicación también tiene lugar (ya que ésta parece estar basada únicamente en el valor transmitido en el campo «a»). No obstante, cuando en la Cuenta B el código JavaScript descarga el listado de ficheros, detecta que la meta-información enviada no se puede descifrar correctamente (dado que las claves no coinciden), por lo que no muestra el fichero pese a que esta información se haya enviado al cliente.
- La deduplicación *cross-user* se realiza en base al resultado de las siguientes operaciones (aquí simplificadas, para facilitar la explicación):

  a = AES (nombre fichero + CRC32(fichero en plano), clave fichero)

  Es decir, si, para dos ficheros se produce el mismo valor «*a*», la plataforma asume que el último en subirse es el mismo que el primero. Al ser «*a*» el resultado de una operación de cifrado con AES, con claves generadas aleatoriamente cada vez, se puede considerar que la probabilidad de provocar una colisión es bastante remota.

- Además, el envío del fichero cifrado se realiza independientemente de si el mismo existe ya en sus servidores. De hecho, el envío del valor «a» sobre el que se basa la deduplicación se hace con posterioridad al envío del propio fichero cifrado. Es decir, **Mega hace deduplicación *cross-user* en cuanto a almacenamiento, pero no en cuanto a comunicación de los datos**, lo que implica que no es posible realizar un análisis como el mostrado en [7].

Por último, por medio del procedimiento explicado anteriormente para reproducir la deduplicación, se comprueba que Mega no puede verificar los hashes (CRC32) recibidos de los ficheros. Esto es debido a que, al estar basado el valor «a» en los datos en plano, no puede reproducir los cálculos a partir de los ficheros cifrados que recibe, para verificar si éstos han sido realizados correctamente.

6.2.2 Almacenamiento

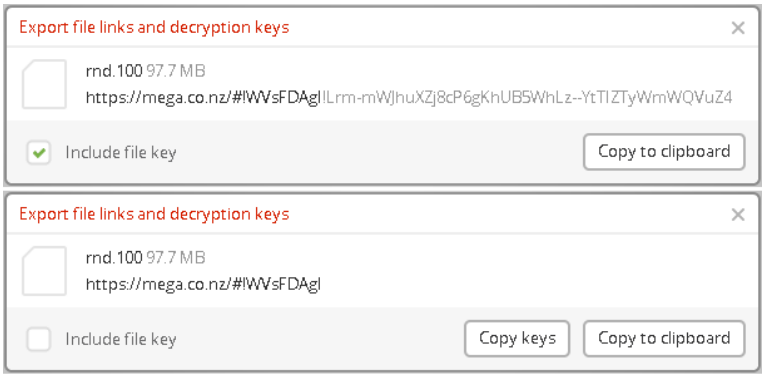
Como se ha visto, los ficheros son cifrados en cliente, utilizando claves AES de 256 bits generadas aleatoriamente. Estas claves son a su vez cifradas con la clave AES maestra del usuario, que se cifra de nuevo a partir de la contraseña del usuario. Por lo tanto, en Mega se puede verificar que los ficheros son almacenados, al menos, con estas garantías.

6.2.3 Compartición

En Mega, es posible compartir ficheros o carpetas a través de links. En este caso hay dos opciones, mostradas en la Figura 30 muestra ambas opciones.

- Incluir la clave de cifrado que da acceso a la información en el propio link. Cualquiera que acceda a este link podrá descargar directamente el fichero o acceder a los contenidos de la carpeta.
- Omitir la clave de cifrado, que se requerirá al acceder al link.

Figura 30. Compartición de ficheros mediante links (con o sin clave).



Alternativamente, se puede enviar el fichero o carpeta directamente a un usuario. En este caso, se cifra la clave AES que se utilizó para cifrar el fichero o carpeta con la clave pública del destinatario, llamando a la función `encryptto`, mostrada en la Figura 31, desde la función `this.copyNodes`.

Figura 31. Función `encryptto`.

```
function encryptto(user,data)
{
    var i, data;
    var pubkey;

    if (pubkey = u_pubkeys[user])
    {
        return crypto_rsaencrypt(data,pubkey);
    }

    return false;
}
```

De las tres posibilidades (compartir enlace con la clave, compartir enlace sin la clave, y

compartir directamente con un contacto), hay que tener cuidado con la opción que incluye la clave en el enlace, ya que cualquier persona que lo consiga (legítima o ilegítimamente), puede acceder directamente a dicha información.

6.2.4 Borrado

Cuando el usuario borra un fichero éste es movido a la papelera de reciclaje. No obstante, Mega no especifica en sus términos de servicio qué hace con un fichero una vez es borrado de la papelera.

6.3 Otros aspectos

Un aspecto importante, dado que la principal funcionalidad criptográfica ejecutada en cliente es a través de las librerías criptográficas descargadas desde los servidores de Mega, es cómo se asegura la integridad de dichas librerías. El proceso se divide en dos fases:

1. Al conectar con https://mega.co.nz, se hace una primera petición al recurso *secureboot.js*, como se muestra en la Figura 32, albergado en el mismo dominio. Como se ha mencionado anteriormente, la conexión con https://mega.co.nz está protegida con SSL, utilizando AES de 256 bits para cifrado.

Figura 32. Peticiones de ficheros JavaScript al cargar Mega.

	Req. Timestamp	Método	URL
1	16/09/14 11:06:41	GET	https://mega.co.nz/
3	16/09/14 11:06:41	GET	https://mega.co.nz/secureboot.js?r=65594115
9	16/09/14 11:06:42	GET	https://eu.static.mega.co.nz/lang/en_22.json
11	16/09/14 11:06:43	GET	https://eu.static.mega.co.nz/sjcl_1.js
12	16/09/14 11:06:43	GET	https://eu.static.mega.co.nz/js/tlvstore_3.js
14	16/09/14 11:06:43	GET	https://eu.static.mega.co.nz/js/crypto_48.js

2. El fichero `secureboot.js` contiene un array de todos los ficheros JavaScript que deben descargarse a continuación, y que almacena los hashes (SHA256) de los mismos. Véase por ejemplo el hash del fichero `crypto_48.js`, a fecha 16/09/2014, mostrado en la Figura 33[43].

[43] El hash se almacena como 8 enteros de 4 bytes cada uno, es decir: 4 enteros x 8 bytes por entero x 8 bits por byte = 256 bits.

Figura 33. Hash del fichero *crypto_48.js* a 16/09/2014.

```
sh1['js/crypto_48.js'] = [-103423678,543269836,1172020738,-345511549,1936661047,832878188,1629233792,1732744625];
```

3. A continuación, se descargan los ficheros JavaScript listados, como se puede observar de nuevo (en parte) en la Figura 32. En este caso, las peticiones se hacen a https://*.static.mega.co.nz, utilizando el mismo tipo de protección que para la conexión con https://mega.co.nz[44].
4. Una vez descargados los ficheros, se comprueban sus hashes con los contenidos en `secureboot.js`, devolviendo error en caso de que no coincidan, como se muestra en la Figura 34.

Figura 34. Comparación de hashes de ficheros JavaScript.

```
if (!nocontentcheck && !cmparrays(sha256(jsl[this.jsi].text),sh1[jsl[this.jsi].f]))
{
    alert('An error occurred while loading MEGA. The file ' + bootstaticpath+jsl[this.jsi].f + ' is corrupt.
    contenterror=1;
}
```

Por lo tanto, la integridad de los ficheros JavaScript descargados al iniciar la sesión, que son los responsables de la protección de toda la información que se almacena en su nube, está garantizada al nivel ofrecido por RSA (de 2048 bits) y AES (de 256 bits).

6.3.1 Gestión personal de claves

Mega no incluye soporte para importar o exportar las claves de usuario generadas desde herramientas externas.

6.3.2 Cambio de contraseña

Teniendo en cuenta la estructura de claves que utiliza el servicio, al hacer un cambio de contraseña hay que actualizar la versión cifrada de la clave AES maestra que se almacena en sus servidores, ya que ésta se almacena cifrada con la contraseña del usuario. Esto es así porque, de lo contrario, su descifrado daría error y no se podrían recuperar los ficheros previamente almacenados. Por tanto, cuando un usuario cambia la contraseña, la clave AES maestra se cifra utilizando (ver Figura 35) la nueva contraseña, y esta nueva versión cifrada se envía de nuevo a los servidores para actualizar la anterior.

[44] Inicialmente, como se dice en https://mega.co.nz/#blog_3, se usaba un certificado de 1024 bits, lo cual generó algo de polémica.

Figura 35. Función para cambio de contraseña en Mega.

```
function changepw(currentpw,newpw,ctx)
{
    var pw_aes = new sjcl.cipher.aes(prepare_key_pw(newpw));

    api_req({ a : 'up',
        currk : a32_to_base64(encrypt_key(new sjcl.cipher.aes(prepare_key_pw(currentpw)),u_k)),
        k : a32_to_base64(encrypt_key(pw_aes,u_k)),
        uh : stringhash(u_attr['email'].toLowerCase(),pw_aes)
    },ctx);
}
```

6.3.3 Críticas a la criptografía con JavaScript

Una de las mayores críticas que ha recibido Mega es el hecho de proporcionar su funcionalidad principalmente a través de librerías criptográficas en JavaScript. Excluyendo los clientes móviles y Sync, para los cuales Mega es una aplicación normal. En el caso de las aplicaciones para Firefox y Chrome, el código JavaScript se descarga como parte de un plugin para dichos navegadores, de manera que no es necesario descargar el código fuente JavaScript cada vez que se accede al cliente web.

Dejando de lado las alternativas soportadas por Mega para distribuir el código, el hecho de utilizar JavaScript para implementar funcionalidad criptográfica ha generado bastante polémica. En concreto, multitud de expertos en el campo son contrarios a esta práctica, que argumentan los siguientes aspectos[45]:

- La distribución del código JavaScript es un punto débil.
- JavaScript no permite por sí mismo la generación de números aleatorios criptográficamente seguros.
- Inmadurez de las librerías criptográficas existentes.
- Maleabilidad del entorno de ejecución.

En cuanto al primer punto, anteriormente se ha descrito el mecanismo para distribuir el código JavaScript en caso de utilizar el servicio tanto a través de alguno de sus plugins como sin ellos. En el caso de no utilizar los plugins, es necesario confiar en la legitimidad del código distribuido por Mega, y al ser transmitido cada vez que se establece conexión (salvo que se haya cacheado), se aumenta el tiempo de exposición a ataques.

En cualquier caso, se establece un mecanismo para la gestión de la descarga de código JavaScript de tal forma que el código transmitido está siempre protegido por TLS. Por lo tanto, siempre y cuando se confíe en que el servicio distribuye código válido (o se audite

[45] Ver http://matasano.com/articles/javascript-cryptography o http://rdist.root.org/2010/11/29/final-post-on-javascript-crypto/ para un análisis algo más detallado de las opiniones contrarias.

cada vez que se descargue, lo cual es imposible en la práctica), además de confiar en la autoridad de certificación que emite el certificado de Mega, este problema parece resuelto de forma satisfactoria.

Por otro lado, en el caso de utilizar los plugins, es necesario confiar tanto en Mega, ya que sigue siendo quien distribuye el código en primera instancia, y en Chrome o Firefox, ya que son quienes gestionan los plugins.

Sobre el problema de la generación de números aleatorios, es sin lugar a dudas una cuestión importante, ya que prácticamente cualquier primitiva criptográfica depende en algún momento de una generación robusta de números aleatorios.

En Mega, esto se resuelve utilizando el movimiento del ratón y la pulsación de teclas como fuentes de entropía para renovar las semillas que se utilizan en el generador de números aleatorios. La Figura 36 muestra un fragmento de la función *mouseMoveEntropy*, en el fichero mouse_6.js, donde se utilizan las coordenadas del cursor para este fin.

Figura 36. Fragmento de la función *mouseMoveEntropy* para el re-establecimiento de la semilla para la generación de números aleatorios basada en eventos de ratón y teclado.

```
var v = ( ( (e.screenX << 8) | (e.screenY & 255) ) << 16 ) | timeValue();

if ( !localStorage.randseed ) {
    if ( bioCounter < 45 ) {
        // `bioCounter` is incremented once per 4 move events in average
        // 45 * 4 = 180 first move events should provide at about 270 bits of entropy
        // (conservative estimation is 1.5 bits of entropy per move event)
        asmCrypto.random.seed( new Uint32Array( [ v ] ) );
    }
```

En cuanto a la tercera crítica, la funcionalidad criptográfica proporcionada está basada en la Stanford Javasript Crypto Library (SJCL[46]), implementada en 2009 por expertos de la Universidad de Stanford. Esta librería es probablemente la librería criptográfica en JavaScript mejor valorada actualmente y desde su creación ha seguido un proceso de mantenimiento activo (ver la gráfica de contribuciones mostrada en la Figura 37, disponible en GitHub).

[46] http://bitwiseshiftleft.github.io/sjcl/

Figura 37. Histórico de *commits* en SJCL.

Obviamente, el código distribuido por Mega, pese a estar basado en SJCL, puede haber sufrido modificaciones. Sobre este aspecto, en la Figura 38 se muestran las diferencias entre la parte de cifrado de una y otra.

La comparación se ha efectuado una vez unificados los estilos de sintaxis, ya que la librería de Mega contiene más *idioms*[47] que la librería oficial, probablemente para minimizar su tamaño. En el caso del cifrado, prácticamente todas las diferencias se corresponden con cambios en el nombre de variables, habiendo también algunos cambios en la organización del código.

Figura 38. Diferencias entre la SJCL oficial y la incluida en Mega.

Por último, la maleabilidad del entorno de ejecución en JavaScript seguramente sea el problema más difícil de atajar, ya que es inherente al propio contexto tecnológico. Un ejemplo de esta problemática es el *bookmarklet* megaPWN[48], un marcador para el navegador que contiene una pequeña porción de código en JavaScript.

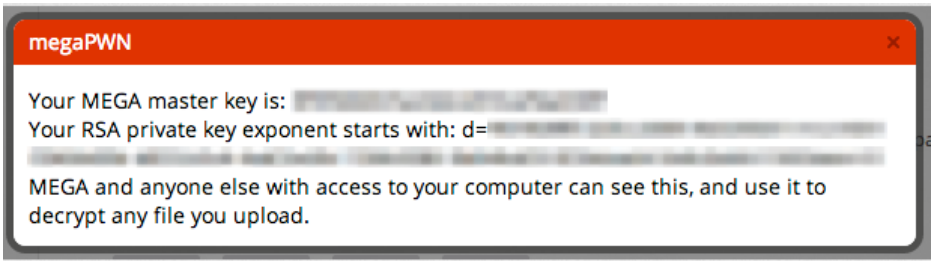
Aunque este programa no parece funcionar en las últimas versiones de los navegadores

[47] http://en.wikipedia.org/wiki/Programming_idiom
[48] http://nzkoz.github.io/MegaPWN/.

web más comunes, en el momento de su publicación permitía acceder a las claves privadas gestionadas por Mega y, según afirma su autor, sin cifrar (ver Figura 39). En este aspecto, la única opción es confiar en el código proporcionado por Mega (o bien auditarlo cada vez que se modifique) y minimizar el riesgo a que componentes externos a Mega modifiquen su comportamiento o vulneren de alguna otra forma la integridad del código que distribuye[49].

Una buena noticia en este aspecto es que todos los navegadores modernos soportan la política del mismo origen (*same-origin policy*[50]), la principal medida para asegurar que un script de un origen determinado no pueda modificar las propiedades cargadas desde otro origen distinto. No obstante, ha habido casos en los que este mecanismo se podía sortear debido a vulnerabilidades en su implementación[51].

Figura 39. Ejemplo de ejecución de megaPWN, extraído de su web.



6.4 Resumen

- **Conexión**: TLS 1.2, con AES de 256 bits en modo CBC y RSA para negociación de clave.
- **Registro**:
  - Como token autenticador, se cifra el email del usuario con una clave AES derivada de la contraseña introducida por el mismo.
  - Se crea una clave AES maestra, que es cifrada con la contraseña del usuario, y un par de claves RSA que se cifran con la clave maestra. Ambas claves se envían cifradas al servidor.
- **Autenticación y *login***:
  - Se regenera el token autenticador derivado inicialmente durante el registro, a partir de la dirección de email y contraseña introducidas por el usuario.

[49] Ver http://www.veracode.com/security/javascript-security para un resumen de vulnerabilidades típicas en JavaScript.
[50] https://code.google.com/p/browsersec/wiki/Part2.
[51] http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html.

  - No soporta autenticación de doble factor.
- **Transmisión de ficheros**:
  - Deduplicación *single-user* en cliente a nivel de fichero.
  - Deduplicación *cross-user* en servidor sobre datos cifrados a nivel de fichero.
- **Almacenamiento de ficheros**: Cifrados utilizando la infraestructura de claves de Mega.
- **Compartición de ficheros**:
  - A través de enlaces, en los que se puede incluir la clave de cifrado o no.
  - A través de contactos.
- **Borrado de ficheros**: No especificado.
- **Otros aspectos**:
  - Las operaciones criptográficas se realizan en cliente (JavaScript).
  - **Compatibilidad con gestión propia de claves**: No.

7 Tabla resumen

En la siguiente tabla se resumen las principales propiedades analizadas en las secciones anteriores. Para detalles específicos sobre cualquiera de ellas, consúltese la sección correspondiente.

Comunicaciones		
Conexión	Dropbox	*TLS 1.2* *AES-GCM 128bits* *ECDHE y RSA*
	Mega	*TLS 1.2* *AES-CBC 256 bits* *RSA*
Registro	Dropbox	*Contraseña enviada por TLS, sin procesar*
	Mega	*Generación token autenticador* *Crea clave AES maestra cifrada con contraseña* *Crea claves RSA cifradas con clave AES maestra*
Login	Dropbox	*Contraseña enviada por TLS, sin procesar* *Doble factor soportado*
	Mega	*Regeneración token autenticador*
Información		
Transferencia	Dropbox	*Des-duplicación single-user en cliente*
	Mega	*Des-duplicación single-user en cliente* *Des-duplicación cross-user en servidor sobre datos cifrados*
Almacenamiento	Dropbox	*Cifrado con clave AES-256 controlada por Dropbox*
	Mega	*Cifrado con clave AES-256 controlada por cliente*
Compartición	Dropbox	*Enlaces no protegidos* *Envío a contactos*
	Mega	*Enlaces (protegidos o no protegidos)* *Envío a contactos*
Borrado	Dropbox	*En servidor tras 30 días*
	Mega	*No especificado*

8 Bibliografía

[1] NIST, «The NIST Definition of Cloud Computing,» *Recommendations of the National Institute of Standards and Technology,* nº 800-145, 2011.

[2] A. Kerckhoffs, «La cryptographie militaire,» *Journal des sciences militaires,* vol. IX, 1883.

[3] NIST, «Special Publication 800-133: Recommendation for Cryptographic Key Generation,» 2012.

[4] NIST, «Anouncing the ADVANCED ENCRYPTION STANDARD (AES),» nº FIPS 197, 2001.

[5] X. Lai y J. L. Massey, «A Proposal for a New Block Encryption Standard,» *EUROCRYPT,* pp. 389-404, 1990.

[6] T. Dierks y E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2,* IETF, 2008.

[7] D. Harnik, B. Pinkas y A. Shulman-Peleg, «Side Channels in Cloud Services: Deduplication in Cloud Storage,» *IEEE Security & Privacy,* vol. 8, nº 6, pp. 40-47, 2010.

[8] J. R. Douceur, A. Adya, W. J. Bolosky y D. Simon, «Reclaiming Space from Duplicate Files in a Serverless Distributed File System,» *ICDCS,* pp. 617-624, 2002.

[9] Jefatura del Estado, «LEY ORGÁNICA 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.,» *BOE,* nº 298, 1999.

[10] Fraunhofer Institute for secure Information Technology;, «On the Security of Cloud Storage Services,» Michael Waidner, Darmstadt, 2012.

9 Enlaces

1. https://www.dropbox.com, accedido por última vez el 26/01/2015.
2. https://mega.co.nz, accedido por última vez el 26/01/2015.
3. http://es.wikipedia.org/wiki/Defense_Advanced_Research_Projects_Agency, accedido por última vez el 26/01/2015.
4. http://www.kurzweilai.net/memorandum-for-members-and-affiliates-of-the-intergalactic-computer-network, accedido por última vez el 26/01/2015.
5. http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals#Publications, accedido por última vez el 15/01/2015.

6. http://thesimplecomputer.info/behind-the-curtain-of-encrypted-cloud-storage, accedido por última vez el 26/01/2015.
7. http://www.engadget.com/2013/03/21/strategy-analytics-cloud-media-market-share/, accedido por última vez el 26/01/2015.
8. https://spideroak.com/, accedido por última vez el 26/01/2015.
9. https://www.wuala.com/, accedido por última vez el 26/01/2015.
10. https://www.owasp.org/index.php/Man-in-the-middle_attack, accedido por última vez el 26/01/2015.
11. http://es.wikipedia.org/wiki/Google_Authenticator, accedido por última vez el 26/01/2015.
12. https://www.incibe.es/technologyForecastingSearch/CERT/Alerta_Temprana/Bitacora_de_ciberseguridad/Celebgate, accedido por última vez el 26/01/2015.
13. https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/autenticacion_passwords_srp, accedido por última vez el 26/01/2015.
14. https://www.bitcasa.com/personal/legal, accedido por última vez el 26/01/2015.
15. http://www.zdnet.com/blog/igeneration/case-study-how-the-usa-patriot-act-can-be-used-to-access-eu-data/8805, accedido por última vez el 26/01/2015.
16. http://www.xatakawindows.com/actualidad-en-redmond/microsoft-se-niega-a-cumplir-una-orden-fiscal-para-exponer-correos-electronicos-almacenados-en-irlanda, accedido por última vez el 26/01/2015.
17. http://www.squid-cache.org/, accedido por última vez el 26/01/2015.
18. http://wiki.squid-cache.org/Features/SslBump, accedido por última vez el 26/01/2015.
19. https://www.wireshark.org/, accedido por última vez el 26/01/2015.
20. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, accedido por última vez el 26/01/2015.
21. https://addons.mozilla.org/es/firefox/addon/firebug/, accedido por última vez el 26/01/2015.
22. https://www.dropbox.com/es/, accedido por última vez el 17/1111/2014.
23. https://www.dropbox.com/static/docs/DropboxFactSheet.pdf, accedido por última vez el 26/01/2015.
24. https://www.dropbox.com/security, accedido por última vez el 09/01/2015.
25. https://www.dropbox.com/privacy, accedido por última vez el 26/01/2015.
26. http://safeharbor.export.gov/list.aspx, accedido por última vez el 26/01/2015.
27. https://www.dropbox.com/help/27, accedido por última vez el 09/01/2015.
28. https://www.dropbox.com/developers, accedido por última vez el 26/01/2015.
29. http://wiki.openssl.org/index.php/Diffie_Hellman, accedido por última vez el 26/01/2015.
30. http://en.wikipedia.org/wiki/Forward_secrecy, accedido por última vez el 26/01/2015.

31. https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/Analisis_trafico_SSL, accedido por última vez el 26/01/2015.
32. https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/autenticacion_passwords, accedido por última vez el 22/01/2015.
33. https://www.dropbox.com/en/help/363, accedido por última vez el 26/01/2015.
34. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), accedido por última vez el 26/01/2015.
35. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#General_Recommendation:_Synchronizer_Token_Pattern, accedido por última vez el 26/01/2015.
36. https://www.dropbox.com/help/227, accedido por última vez el 26/01/2015.
37. https://www.dropbox.com/help/25, accedido por última vez el 26/01/2015.
38. https://www.dropbox.com/help/296, accedido por última vez el 26/01/2015.
39. https://www.dropbox.com/en/help/28, accedido por última vez el 26/01/2015.
40. http://www.washingtonpost.com/blogs/the-switch/wp/2013/09/06/kim-dotcom-is-still-wanted-by-the-fbi-but-that-isnt-slowing-him-down, accedido por última vez el 26/01/2015.
41. https://mega.co.nz/#help, accedido por última vez el 26/01/2015.
42. http://tools.ietf.org/html/rfc3610, accedido por última vez el 26/01/2015.
43. https://mega.co.nz/#blog_3, accedido por última vez el 26/01/2015.
44. http://matasano.com/articles/javascript-cryptography/, accedido por última vez el 26/01/2015.
45. http://rdist.root.org/2010/11/29/final-post-on-javascript-crypto/, accedido por última vez el 26/01/2015.
46. http://bitwiseshiftleft.github.io/sjcl/, accedido por última vez el 26/01/2015.
47. http://en.wikipedia.org/wiki/Programming_idiom, accedido por última vez el 26/01/2015.
48. http://nzkoz.github.io/MegaPWN/, accedido por última vez el 26/01/2015.
49. http://www.veracode.com/security/javascript-security, accedido por última vez el 26/01/2015.
50. https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy, accedido por última vez el 26/01/2015.
51. http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html, accedido por última vez el 26/01/2015.

10 Tabla de figuras

ANEXO 1 – CIFRADO CONVERGENTE

Los algoritmos de cifrado convergente [8] se utilizan para producir siempre un mismo texto cifrado a partir de un mismo texto plano, de forma que sólo quienes conozcan el fichero plano original puedan recuperarlo. Aunque esto implica una seguridad menor en cuanto a la privacidad de la información, ya que no garantizan la «*indistinguibilidad*» de los textos cifrados, es una útil primitiva para aumentar la eficiencia de sistemas de almacenamiento públicos (o privados de gran escala). Para ello, combinan hashes criptográficos con algoritmos de cifrado simétricos y asimétricos. En la Figura 40 se muestra el proceso de cifrado convergente definido en [8]. Los datos se cifran mediante un algoritmo de cifrado simétrico, utilizando como clave el hash de los propios datos. Esta clave es a su vez cifrada con la clave pública del usuario, y el par formado por los datos cifrados y la clave cifrada es almacenado en el servidor.

Cuando se añaden nuevos datos al servidor, si ya existen, sólo se repite el proceso de cifrado de la clave, pero en este caso utilizando la clave pública del nuevo usuario. Al servidor se sube únicamente la nueva clave cifrada, que se relaciona de alguna manera a los datos cifrados (por ejemplo, actualizando un listado de claves asociadas). De esta forma, cuando un usuario quiera acceder a la información, tendrá que descifrar la clave simétrica adecuada utilizando su clave pública, por lo que únicamente los usuarios que hubieran ejecutado el proceso anterior podrán obtener la clave simétrica correcta y finalmente descifrar los datos.

No obstante, cabe destacar que los clientes de las aplicaciones de almacenamiento en la nube no siempre poseen pares de claves pública-privada, por lo que los esquemas de cifrado convergente en dichos casos pueden variar sobre lo explicado anteriormente. Por ejemplo, el listado de claves simétricas cifradas (cada una con una clave pública de un usuario distinto) podría ser sustituido por un listado cifrado, con una clave bajo el control único del proveedor del servicio, de los usuarios que poseen el fichero asociado.

Figura 40. Cifrado convergente en el contexto de almacenamiento en la nube.