Security in cloud storage services
Analysis of Dropbox and Mega
Author

Jesús Díaz Vico

This study has been carried out with the collaboration of David Cantón Araujo, Daniel Fírvida Pereira, Asier Martínez Retenaga and Antonio Rodríguez Fernández.
INDEX

1 ABOUT THIS STUDY ........................................................................................................ 5

2 INTRODUCTION ........................................................................................................... 6

3 ASPECTS WORTH CONSIDERATION ........................................................................... 9
   3.1 Management and use of encryption ........................................................................ 9
   3.2 Authentication ........................................................................................................ 10
   3.3 Data deduplication ................................................................................................. 10
      3.3.1 Client side deduplication vs server side deduplication .................................. 11
      3.3.2 Single-user deduplication vs cross-user deduplication .................................. 12
      3.3.3 File deduplication or block deduplication ....................................................... 13
   3.4 Data sharing between users .................................................................................. 14
   3.5 Data deletion .......................................................................................................... 14
   3.6 Types of clients ..................................................................................................... 14
   3.7 Other factors ........................................................................................................ 15

4 ANALYSIS METHODOLOGIES .................................................................................. 16

5 DROPBOX ANALYSIS ................................................................................................. 18
   5.1 Communications .................................................................................................... 18
      5.1.1 Connection ...................................................................................................... 18
      5.1.2 Registration .................................................................................................... 19
      5.1.3 Login .............................................................................................................. 20
   5.2 Information management ....................................................................................... 21
      5.2.1 Transmission .................................................................................................. 21
      5.2.2 Storage .......................................................................................................... 24
      5.2.3 Sharing ......................................................................................................... 24
      5.2.4 Deletion ....................................................................................................... 24
   5.3 Other aspects ......................................................................................................... 25
      5.3.1 Personal management of keys ....................................................................... 26
5.4 Summary ............................................................................................................. 26

6  ANALYSIS OF MEGA .................................................................................. 27

6.1 Communications .................................................................................................. 27
   6.1.1 Connection ....................................................................................................... 27
   6.1.2 Registration ...................................................................................................... 28
   6.1.3 Login ................................................................................................................. 30

6.2 Information management .................................................................................... 32
   6.2.1 Transmission .................................................................................................... 32
   6.2.2 Storage ............................................................................................................. 37
   6.2.3 Sharing ............................................................................................................. 37
   6.2.4 Deletion ............................................................................................................ 39

6.3 Other aspects ....................................................................................................... 39
   6.3.1 Personal management of passwords ............................................................... 40
   6.3.2 Change of password ......................................................................................... 40
   6.3.3 Criticism of JavaScript-based cryptography ..................................................... 41

6.4 Summary ............................................................................................................. 44

7  SUMMARY TABLE .......................................................................................... 46

8  BIBLIOGRAPHY .......................................................................................... 47

9  LINKS ............................................................................................................. 47

10 GRAPHICS TABLE ....................................................................................... 49
1 About this study

Cloud services are increasingly popular, with an ever-growing number of users. One of the most common services is applications used for cloud storage. This study picks up on different aspects related to these services and their security. The study is divided in two main blocks:

- In the first part, up to and including section 3, the main aspects that can affect the security of cloud storage solutions are introduced, mainly from a technical perspective, but also mentioning other important aspects. In particular, a lot of emphasis is put on encryption and authentication, deduplication policies and the sharing of data between users. The main alternatives and the implication that each of these points can have on the security and privacy of users are described.
- In the second part of the study, from section 4 onwards, a common framework is established to analyse cloud storage applications. This is employed in sections 5 and 6, analysing the security of two of the most commonly used applications in the market which are available for any user. The applications that have been analysed, Dropbox [1] and Mega [2], are representative since the former is probably the most well-known application and the latter, besides being a popular application, puts a special emphasis on security. The measures incorporated by these two applications for the aspects described in the first part of the study have been verified.

It is possible to assess which solutions adapt better to the specific needs that may arise with the principles displayed in the first part of this study. Besides being aimed at two of the most popular applications, the analysis included in the second part can also serve as a framework to be reproduced with other applications or expanded depending on the specific requirements.

Finally, it is important to keep in mind that the tests [3] carried out during this study took place in the last trimester of 2014. Therefore, the obtained results may vary if they are reproduced at a later date.

[1] https://www.dropbox.com/
[3] All the tests carried out for this study were done so with test accounts, created specifically for this purpose.
2 Introduction

The concept of a cloud of global electronic services started to take shape with researchers such as Joseph C.R. Licklider, who already contemplated the advantages of a network infrastructure that enabled the possibility of sharing services in a memorandum for ARPA\(^4\) in 1963. Furthermore, he predicted that one of the greatest difficulties that this infrastructure would encounter would be its compatibility with different programming languages and communications protocols.

This idea evolved into ARPANet, which in turn was the origin of what is today known as the Internet. Over time the Internet has extended to all fields of companies and people, and the typology of products and services designed to share information has evolved accordingly to needs of new users. Therefore, not only products such as the grid-plotting tool are offered which Licklider made reference to, but also common-use utilities such as storage, web servers, email systems, etc.

The evolution and constant improvement of the technology that is available has notably influenced this aspect: the increase in bandwidth and development of international standards for the interoperability of different devices has facilitated the provision of services from what has come to be known as the cloud.

Nowadays, these incidents are provoking a lot of activities that until recently were done «locally» or in private corporate infrastructures to be diverted to the cloud, such as listening to music (e.g. Spotify), watching videos (e.g. Youtube), sending emails (e.g. Gmail) or storage (e.g. Dropbox).

These services provide the user with a better experience, as they avoid having to carry out costly or complicated installations and set-up and maintenances tasks, enabling the user to only worry about his everyday or professional tasks, incurring also in fewer costs on many occasions. Besides, the final user also acquires important advantages, such as a greater availability of the service.

Like every science or technology, over time, concepts have matured to help understand and improve this new environment. Therefore, as explained in «The NIST Definition of Cloud Computing» [1], Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), which can be used by office or web clients, are included within the service models offered in the cloud.

SaaS specifically refers to the use of software available from the cloud (e.g. Spotify, YouTube, Gmail, Dropbox); PaaS to the use of a remote calculation platform (e.g. operating systems configured with development settings, databases, web servers); and IaaS to the use of delocalized infrastructures (e.g. virtual machines and storage systems).

\(^5\) It specifically illustrates an example of its idea through a grid-plotting program that can be shared on a network of 4-8 computers. Source: http://www.kurzweilai.net/memorandum-for-members-and-affiliates-of-the-intergalactic-computer-network.
It must be noted that it is also possible to establish inter-relations between different service suppliers. Therefore, for example, an IaaS supplier can offer virtual machines to a PaaS supplier that assembles XAMP servers, which, at the same time could be contracted by an online TV broadcaster on the web (in other words, an SaaS supplier). Obviously, the three links do not have to be present in the same chain.

One of the most popular services in recent years is undoubtedly cloud storage. This study will analyse the security of two of the most popular applications: Dropbox, perhaps the most well-known and commonly used of all, and Mega, which is also quite popular and has focused greatly on security since its creation.

As shown on Graphic 1, on average, 21% of citizens in the European Union (EU-28) used a cloud storage system in 2014 to store files, and 15% used them to share. In some countries (Denmark, United Kingdom, Luxembourg, Sweden and Holland), they surpass 30% and 20% respectively (in some cases by a significant amount).

Figure 1. Use of cloud storage systems in European Union countries.

Dropbox was probably the first application of this kind to expand globally, with its first version dating back to 2008. Since then, numerous alternatives have appeared with more or less success, many of which are available today.

---

6 http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals#Publications
7 See for example http://thesimplecomputer.info/behind-the-curtain-of-encrypted-cloud-storage/ for a complete list
The majority of the market share is currently spread out between a few services, as can be deduced from the market study carried out by Strategy Analytics in the last third of 2013, which interviewed 2300 American citizens (see Graphic 2). As can be observed, iCloud, Dropbox and Amazon Cloud Service amassed approximately 50% of the market share towards the end of 2013, followed by Google Drive with 10% of the share.

Likewise, other applications where security is prioritized can be found although perhaps at the expense of other characteristics. Amongst these kinds of options the most well known are probably Mega, SpiderOak\(^9\) and Wuala\(^10\).

The objective of this document is to analyse the security and privacy characteristics of Dropbox and Mega, two of the main cloud storage applications that are available for free on the market. The main concepts that will influence the analysis and that are used as a base in this study are exposed in the document.

---

\(^8\) [http://www.engadget.com/2013/03/21/strategy-analytics-cloud-media-market-share/](http://www.engadget.com/2013/03/21/strategy-analytics-cloud-media-market-share/)

\(^9\) [https://spideroak.com/](https://spideroak.com/)

\(^10\) [http://www.wuala.com/](http://www.wuala.com/)
3 Aspects worth consideration

It is necessary to understand how cloud storage systems work in order to establish a solid base that enables the understanding of the factors that affect their security. Following is a summary of the main aspects that affect the level of privacy offered to the end-user.

3.1 Management and use of encryption

One of the cornerstones of the security of every cryptographic function is the process of generating passwords, as Kerckoffs already observed in 1883 [2]. In that sense, it is recommendable to adhere to national or international recommendations such as NIST’s Special Publication 800-133 [3]. But not only is the derivation algorithm or generation of passwords important, so is where they are generated and how they are managed afterwards.

There are applications that generate passwords for encryption in servers, but that to send information use another password or even another type of encryption (for communications, normally SSL/TLS is used). Therefore, even if the data that is sent out is encrypted, the server will have access to it, so the privacy guarantees of the user decrease.

Some applications also enable the user to generate his passwords using external tools, and then subsequently import them with the cloud storage application’s client. In this aspect, this is the most robust option as long as the external tool also follows the password generation recommendations, since it allows the user to choose tools that have been designed specifically for this and which the user already trusts (for example, through a previous personal experience).

Obviously, the management of data encryption is a fundamental aspect to guarantee its confidentiality. In this aspect, there are various points worth acknowledging:

1. The encryption algorithm itself: once again, the use of algorithms that are verified by the scientific and technological community is essential. Standards such as AES [4] or IDEA [5] tend to be good alternatives adopted by many suppliers.
2. Just like it is important to know where the cryptographic passwords are generated, so is knowing where (client or server), and why (transmission or storage), the data is encrypted. For example, if the data is only encrypted
during its transmission, but is then stored in clear in the servers, there is still a threat to the privacy of the users.

3. **What password the data is encrypted with:** if it is encrypted with a password that is created and managed by the service’s supplier, the user «loses» the effective control of its data once it is transmitted to the servers. If it is encrypted with a password created by the user (and if it is done so with an external application, that is better), the user will continue to control the data even when it has been sent to the server.

### 3.2 Authentication

Like in every communication, it is important for both ends to authenticate each other, as well as the information they transmit. On one side, **the servers must always authenticate** before starting a communication, using digital certificates emitted by trustworthy authorities. As can be observed in the definition of the TLS protocol [6, p. 90], as long as the server is authenticated, the transmission will be secure from **man-in-the-middle** type attacks.

The authentication of the client is more delicate, since demanding that it be done through a digital certificate can be too harsh a request (from a usability point of view).

The most common authentication mechanism in all cloud storage services is the use of passwords, although more and more services have incorporated multifactor authentication mechanisms, such as Google Authenticator[^12].

### 3.3 Data deduplication

The concept of data deduplication refers to the management that the service’s supplier carries out of the data received on two or more occasions from the user himself or different users. In this case, the server can store the new data or simply create a link with the already existing data.

Although this is not an element that directly affects users, it is important, as it enables a more efficient management of the bandwidth and storage systems (since it will be possible to store more files or blocks using fewer bytes). Besides, this probably has a direct influence in the price that the end-user pays, given that a more efficient management of the communication and storage implies fewer costs for the server and, at the same time, results in a more economical service.

This is not only important in economic or effective terms, but it can also affect the privacy of the system. To understand the reasons for this, following is an explanation of the different data deduplication alternatives.

### 3.3.1 Client side deduplication vs server side deduplication

As can be deduced from the name, deduplication can be carried out from the server or directly from the client. In a server side deduplication, the client sends the data to the server and it is the server who executes the deduplication algorithms.

In a client side deduplication, the client normally applies a hash function on the block or file and sends the result to the server. The server receives it and checks if it already has a copy of this data, so the data is only sent if the server does not possess a copy of the file. Both actions are shown in a schematic form in Graphic 3 and Graphic 4.

---

**Figure 3. Server side deduplication.**

---

**Figure 4. Client side deduplication.**

---

If carried out from the server, there is a larger cost of communications, and if the data
has not been previously encrypted by the user (with his personal password), it can also violate his privacy. On the other hand, if it is carried out from the client, the user obtains information regarding whether the file already exists in the system or not (known as cross-user deduplication, explained further ahead). The attacker could use this information to extract information, as observed in Harnik et al’s study [7].

### 3.3.2 Single-user deduplication vs cross-user deduplication

In single-user deduplication, only duplicities in the data of a sole user are checked. On the contrary, in cross-user deduplication, this verification is carried out in all the system’s users. In Graphic 5 a single-user deduplication scheme is shown and in Graphic 6 the equivalent for cross-user deduplication.

For simplicity reasons, only the server side deduplication case is shown (for a client side deduplication the verification shown in the graphs would follow the same principle, but sending the hash instead of the file).

**Figure 5. Single-user deduplication.**

![Single-user deduplication diagram](image)

**Figure 6. Cross-user deduplication.**

![Cross-user deduplication diagram](image)

With single-user deduplication it is easier to preserve the privacy of the user’s
information, as no verifications have to be carried out between the different users and it is more compatible with the encryption algorithms. For example, when different users have different passwords, it normally makes no difference if the data is encrypted in the client, and it can be carried out indifferently in the client or the server. The down side, logically, is that the communication and storage costs increase.

Cross-user deduplication is therefore more efficient in terms of costs, as it enables greater savings of transmitted and stored bytes. However, this is the origin of various complications.

- On one hand, as mentioned previously, when the cross-user deduplication is carried out in the client, it enables the attacker to obtain information regarding which data other users store.
- On the other hand, supposing that (ideally) each user has his own password and that any file is encrypted before being sent to the server, if \( N \) users upload the same file to the cloud, \( N \) different copies will be produced for a same file. Therefore, the deduplication of the file is not possible, or at least following the traditional solution.

The most direct form of solving this problem, adopted by some suppliers, is using one common password (controlled by the supplier) for every client. But this deprives the users from controlling their data, precisely because the password used is not known only by them.

A common technique that partially solves this problem is using convergent encryption algorithms (see ANNEX 1 – Convergent encryption for a summary of this type of encryption, or the original proposition [8] for further details). Parting from a plain text, this type of encryption produces the same text in encrypted form, meaning that the encrypted text can only be decrypted by people who know the original plain text. This is compatible with every user having a different personal password, but it still allows the server to know what users have the same files, although it does not have access to more information.

### 3.3.3 File deduplication or block deduplication

This variation indicates the depth at which the deduplication is carried out. If it is carried out at a file level, the identical files will simply be de-duplicated. Alternatively, when the deduplication is at a block level, the files are divided into blocks (normally of a predetermined size) and the same verification for each block formed with the files is carried out.

If a cross-user deduplication is carried out at a file level it can be deduced that another user has exactly the same file (if the deduplication is calculated on the non-encrypted
files). If the deduplication is carried out at a block level instead of a file level, it can be used as a base to carry out an attack that adequately combines brute force with the manipulated files, with the aim of illegitimately obtaining specific parts of a file, as indicated in Section 2.2 of Harnik et al’s study [7].

3.4 Data sharing between users

Some methods, although they’re not unsecure themselves, can have implications in the security and privacy of users. For example, if files are shared through a link (function offered by many services), anyone who has access to this link, or can somehow predict it, can access the associated information unless if it is encrypted. In other cases, the file can be directly shared through the contacts that are configured in the service, in other words, without generating a public link.

3.5 Data deletion

Typical behaviour in storage suppliers is to keep a file for a predetermined number of days after it has been deleted and before definitively deleting it.

Other policies can affect this general behaviour. For example, if a deduplication of files has occurred between users (cross-user), the counter of the number of users who possess an already existing file in the system increases, and a reference of it is included. Therefore, even if the user deletes his file, if other users also have it in their account, the real data is not deleted, only the reference is.

Finally, when deleting a file (not a reference), what kind of deletion is carried out? Is it a safe deletion (with multiple overwritings) or is this file or block simply deleted? There is probably no way of determining which of these options applies to each supplier, so users must trust that they manage the deletion adequately, depending on what they declare in the conditions of their service.

3.6 Types of clients

Ideally, the different clients that are implemented by a supplier must be equivalent in terms of their functionality, although this is not always the case. This is mainly down to factors such as cutting down on computing or communications costs (important in mobiles for example), or available APIs which are normally limited on web browsers. Sometimes security is sacrificed in favour of achieving better results in the aforementioned aspects, or in the final usability of the application. For example, the application Bitcasa specifies this in the security section of its legal information page13:

---

13https://www.bitcasa.com/legal
As a result, it is important to keep this factor in mind, especially if different clients are going to be used.

### 3.7 Other factors

Besides the aforementioned aspects, cloud storage suppliers incorporate other mechanisms to increase the efficiency of the system. Still, there is no evidence that these factors have a direct influence in the system’s general security (unless if, evidently, there are vulnerabilities in its implementation that can activate an attack). Following is a list of these mechanisms:

1. Encoding based on differences (generally known as delta encoding). It consists in only sending the differences of a file with the last stored version, with the aim of creating more efficient communications.
2. Compression. As the word indicates, it consists in compressing the information before sending it.

Obviously, it must also be mentioned that the different suppliers can vary the output put at the disposal of users: storage capacity, subscription costs depending on the type of account, specific functions (format adaptation, etc.). Although these are evidently important, this study will not go into greater detail because they are not directly related to security.

On a legal terrain, it is important to know the rules regarding these cloud storage services. This gives an idea of the backing the authorities offer or when they can be legally authorized to access the uploaded information.

In Spain, the law that regulates the treatment of personal data is the LOPD [9]. However, given that the majority of services on the cloud are not located in Spain, it is important to also know foreign laws. In the European Economic Area framework, the 95/46/EC directive from 1995 about data protection establishes limits in the treatment of personal data. In the USA [10], it is important to acknowledge the USA PATRIOT ACT, created after the 9/11 terrorist attacks in the city of New York; and the third-party doctrine from the Fourth Amendment, aimed at protecting the privacy of users. The Safe Harbor Framework is also relevant. This is a cooperation agreement between the USA and the European Union to ensure the secure transmission of information between both countries.
4 Analysis methodologies

Important aspects that could influence the security of cloud storage applications were introduced in the previous section. However, it is important to focus on the concrete actions that can be executed in these applications when undergoing an analysis. In this sense, some of the aspects that have been touched upon are actions that are directly possible in an application (such as sharing and deleting files), whereas others are more transverse properties (such as encryption and authentication). Following are some basic guidelines on how to analyse a cloud storage application with these properties.

The main operations can be divided in two parts:

- Those related to implemented communications protocols. Specifically, the type of connection used, the registration process for new users and how they login subsequently is relevant. The encryption and authentication mechanisms are especially important (see sections 3.1 and 3.2) in these operations.
- Those related to the management of information (files and folders), given that this is the main use of these applications. How the information is transmitted is important in particular, with the encryption and authentication mechanisms also being essential (sections 3.1 and 3.2) but so is the execution or non-execution of a deduplication, as explained in section 3.3. How the information is stored once it reaches the company’s servers, if it is encrypted or not, and how. How the information is shared with other users and the management of the deleted information that has been uploaded is also worth acknowledging, as explained in sections 3.4 and 3.5.

The types of clients (section 3.6) that support the application also have to be acknowledged, executing a separate analysis of each one which presents varying behaviours in the aforementioned operations. And, of course, each specific application may require acknowledging additional aspects, such as the manner of implementing cryptographic functions (a conflictive aspect in Mega for example, as explained in section 6.3.3).

Different cloud storage applications can be analysed using a common framework which is summed up in Graphic 7.
In the following sections, this methodology is applied to analyse the specific cases of web clients for Dropbox and Mega. However, many of the aspects treated are common to all their clients. In these concrete cases, the tools that have been used for the analysis are:

- **Squid**\(^{14}\), with the **SSL Bump**\(^{15}\) extension, acting as the *Man In the Middle*.
- **Wireshark**\(^{16}\) and the OWASP **ZAP**\(^{17}\) utility to analyse traffic.
- **Firebug**\(^{18}\) to debug web pages.

The functioning of these aspects is compared, along with the management of the information they transmit, using the traffic that is captured and the information that is obtained through debugging applications as a starting point.

---

\(^{14}\) [http://www.squid-cache.org/](http://www.squid-cache.org/)

\(^{15}\) [http://wiki.squid-cache.org/Features/SslBump](http://wiki.squid-cache.org/Features/SslBump)

\(^{16}\) [https://www.wireshark.org/](https://www.wireshark.org/)


\(^{18}\) [https://addons.mozilla.org/es/firefox/addon/firebug/](https://addons.mozilla.org/es/firefox/addon/firebug/)
5 Dropbox analysis

Dropbox offers free accounts with a capacity of 2 GB (and up to an additional 16 GB through invitations) and Pro accounts of 1 TB. It includes a support for Windows, Mac, Linux, Apple devices, Android and Blackberry. It uses delta encoding and enables users to share folders with other users, as well as create public links for specific files.

This service registers a history of the changes made during the last month, allowing users to recover older versions and even deleted files. Regarding security, it **protects communications with the SSL protocol and encrypts the data in the servers with AES-256**. In the security clauses, it also specifies that not every mobile client is compatible with the transmission of ciphered data, so **there could be instances in which certain information is not encrypted before it is sent** (emphasizing on multimedia content).

On another note, despite the fact that Dropbox is adhered to the SAFE Port Act and that they put a lot of emphasis on the fact that staff members can only access data under exceptional conditions, Dropbox has access to the information stored by its users in the cloud. They specifically indicate that they can decrypt and pass on stored information if the law requires it.

Various APIs that enable users to interact with different components of the service and develop their own applications are also available.

5.1 Communications

5.1.1 Connection

128 bit AES encryption is used both for web connections (see Graphic 8) and connections with desktop applications, and Diffie-Hellman key exchange, authenticated with RSA, is used as a password negotiation algorithm. This means that:

- The symmetric key is generated with the Diffie-Hellman key exchange algorithm, in other words, using new public values in each negotiation instead

---

19 https://www.dropbox.com/
20 https://www.dropbox.com/security, section “Protect files in transit and at rest”.
of fixed values within a digital certificate. This is how Forward Secrecy\textsuperscript{25} is achieved.

- The server uses its digital identity, generated with RSA, to authenticate the negotiation and avoid Man in the Middle type attacks.

Figure 8. SSL connection with Dropbox.

5.1.2 Registration

To create a Dropbox account from its website, the following fields must be introduced: name, surname, email address and password. These values are transmitted to the server (via a TLS connection as shown previously). It is noteworthy that the password, despite being sent through a TLS connection, is received by Dropbox without being processed by a hash function or similar, with the servers thus having access to the clear-text password in. This can be observed in Graphic 9, where a screenshot of the TLS traffic (already decrypted) of a registration process can be seen, obtained with Wireshark, using Squid\textsuperscript{26} to intercept the exchanged information.

\textsuperscript{25}http://en.wikipedia.org/wiki/Forward_secrecy.

\textsuperscript{26}http://www.inteco.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/Analisis_trafico_SSL.
Therefore, despite the communication being protected with TLS, Dropbox receives the password itself, instead of a hash or any other function of it. This is not a good practice, because if an attacker compromises Dropbox’s, he could directly obtain the passwords of new users, regardless of whether these are subsequently encrypted for their storage. There are various alternatives to improve this process\textsuperscript{27}. For example, the impact of compromises to the server could be limited using salted hashes from the password. Alternatively, this process would be notably safer if SRP\textsuperscript{28} or similar protocols were used.

5.1.3 Login

The basic authentication is based on user and password pairs. In the web application, just like in the registration process, the password is sent via TLS, but without being processed previously with a hash function or similar functions, with the servers having access to the password in clear. Therefore, an attacker that compromises Dropbox’s servers can have access to passwords in clear of users accessing their accounts.

Dropbox also enables a multifactor authentication\textsuperscript{29}, which adds additional authentication mechanisms besides the password. This option can be managed from the configuration options on the website, on the security tab. There are two existing

\textsuperscript{27}https://www.incibe.es/blogs/post/Security/SecurityBlog/Article_and_comments/password_based_authentication
\textsuperscript{28}https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/autenticacion_passwords_srp
\textsuperscript{29}https://www.dropbox.com/en/help/363
alternatives to this:

- Indicating a mobile phone number, where an SMS is sent every time the user needs to be authenticated.
- Using a mobile application\(^{30}\), where a QR code is initially scanned during configuration to establish the seeds that will later be used to generate temporary random numbers. When the user has to authenticate himself, he must introduce the random number shown in the application.

As a mechanism for protection from \(\text{CSRF}\)\(^{31}\) attacks, Dropbox establishes a session token that is different after a satisfactory authentication, following OWASP’s recommended synchronizer token pattern\(^{32}\) to avoid this type of attacks. This 18-byte token is subsequently transmitted in every request.

### 5.2 Information management

#### 5.2.1 Transmission

As explained in Section 3.3, an important aspect that can affect the privacy of users is whether deduplication is executed or not between users (cross-user) when sending information to the cloud service. When transmitting files in Dropbox, these are cut up by default into 8MB fragments if the web interface is used and 4MB fragments if Python’s API is used, according to the definition of the `upload_chunked` method in the `client.py` file:

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

To verify if deduplication occurs in this case, the methodology described in the work by Harnik et al\(^{7}\) has been followed. In other words, this implies verifying the amount of data transmitted and the total time of the transmission for different files from the accounts of different users. The tests carried out have been specifically executed in the following manner:

1. A large random file has been created (`rnd.100` in the following example) to minimize the probability that it may already exist in a user’s Dropbox account.

---

\(^{30}\) For example, Google Authenticator.

\(^{31}\) [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)).

For example, random files can be generated in Linux in the following way (with a size of 100 MB in this case):

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

It must be noted that although it specifies that the quality of the random numbers generated is low (parameter 0), in this case it is not important, since they are not going to be used in cryptographic processes.

2. Subsequently, the `rnd.100` file is copied to the Dropbox folder of an account A, measuring the amount of bytes transmitted during the synchronization with Dropbox’s servers. As shown in Graphic 10, from all the conversations captured by Wireshark, the highlighted conversation corresponds with the session sending the file. Approximately 122MB have been transmitted in it.

![Figure 10. Number of bytes transmitted from uploading the rnd.100 file to the A account.](image)

3. The `rnd.100` file is then copied to the Dropbox folder of a B account, measuring the new quantity of bytes transmitted. Once again, in Graphic 11, it can be observed that the amount of bytes transmitted is practically the same. This is an indication that a **cross-user deduplication is not executed in the client** (not on a block level or file level).
Following the same procedure (with a different rnd.100.2 file), but for only one user account, it is demonstrated that Dropbox does perform single-user deduplication, as shown on Graphic 12 and Graphic 13. It can be seen that the first time that the file is uploaded (Graphic 12) approximately 110 MB are sent, whereas in the second delivery (Graphic 13), only around 0.02 MB are sent.
5.2.2 Storage
According to the company\textsuperscript{33}, the information uploaded to the cloud is encrypted with 256 bits of AES when it is stored. They also confirmed\textsuperscript{34} that only a small number of employees have access to the keys used to encrypt the information.

5.2.3 Sharing
Both desktop client and web clients can share files or directories by generating a link. The generated links have the following form:

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

Normally, anyone who has the link will have access to the file or directory, although in paid accounts it is possible to establish visibility restrictions, by establishing a password or setting an expiry date for the link.

5.2.4 Deletion
According to Dropbox\textsuperscript{35}, deleted files are maintained for 30 days in standard accounts, and can be recovered in this interval from the website’s interface. For accounts that include the Extended Version History feature, this period increases to one year.

\textsuperscript{33} https://www.dropbox.com/security.
\textsuperscript{34} https://www.dropbox.com/help/27.
\textsuperscript{35} https://www.dropbox.com/help/296.
On the desktop application, Dropbox maintains a concealed directory, `.dropbox.cache`, where the deleted files are stored in subfolders that are organized by date. In this case, if the account is configured on various devices, by deleting a file on one device, the rest are updated as a consequence. For example, if the `randomtext.txt` file has been deleted on the 2nd of May of 2014, the folder created will be named 2014-05-02, with similar content to that shown on Figure 14.

![Figure 14. Deleted files in the Dropbox desktop client.](image)

### 5.3 Other aspects

Apart from the previously mentioned properties which are related to the form in which Dropbox protects the information of its users, *at rest* and in transit, there are also other notable security measures. For example:

- In mobile devices, Dropbox enables the creation of personal passcodes\(^{36}\), which adds a security layer to the code or an unblocking pattern which could be active in the device.
- All the linked devices\(^{37}\) can be managed from the web interface. Therefore, if an individual loses a smart phone with Dropbox installed, he can unlink the

\(^{36}\) [https://www.dropbox.com/help/227](https://www.dropbox.com/help/227)

account remotely to deny access from the missing phone and therefore protect the confidentiality of the files.

5.3.1 Personal management of keys

To be able to create personal keys from external software and subsequently import them on Dropbox would be a notable improvement for users who already have their own keys. Besides, it would offer additional guarantees to protect the user’s data at rest, since it would be encrypted with the user’s key, instead of with a key that is not under his control. Currently, Dropbox does not have this function, although they do not rule out adding it in the future.38

5.4 Summary

Based on what has been shown in this study, the main characteristics that can influence the security of the service provided by Dropbox are:

- **Connection**: TLS 1.2, 128 bytes of AES in GCM, ECDHE and RSA mode for key negotiation.
- **Registration**: The password is sent through TLS but without processing through a hash function or similar, with the servers having access to the password in clear.
- **Authentication and login**:
  - The password is sent through TLS but without processing through a hash function or similar, with the servers having access to the password in clear.
  - Support for double factor authentication.
- **File transmission**:
  - Executes single-user client side deduplication, at a file level.
  - Does not execute client side cross-user deduplication.
  - Server side deduplication could not be verified.
- **File storage**: Encrypted with 256 bits AES. Keys controlled by Dropbox.
- **File sharing**:
  - Through links. Free clients cannot protect them with a password or in any other way.
  - Through contacts.
- **File deletion**: According to its service terms for free clients, files are deleted from the server 30 days after the user deletes them.
- **Other aspects**:

---

6 Analysis of Mega

Mega\(^{39}\) offers a web client, affirming that it works with any modern browser, although they advise using Google Chrome, Mozilla Firefox or Opera Next \(^{40}\). There are also mobile clients for Android, Blackberry and iOSs and Sync clients (for desktop synchronization) compatible with Linux, Mac and Windows. Additionally, it provides an API, which enables the creation of non-official clients. For free accounts, the storage limit is 50 GB, whereas for paid accounts it varies from 500 GB to 4 TB, along with growing available bandwidths.

One of the main attractions of Mega is that it includes the privacy of its users in its design (“privacy by design”). The most notable aspect is that the cryptographic functionality aimed at protecting the information stored in Mega’s servers is executed in the client’s computer through JavaScript code downloaded from its servers. Therefore, Mega confirms that unless a user makes the link to a file that is stored in their servers public, along with the key used to encrypt it, they will not have any way to access the content.

**Note**: Some of the following sections include references to files with Mega’s source code. The name of these files tends to follow the `<name>_<number>.js` format where `<number>` is a whole number that increases when Mega introduces changes in the file. Therefore, some of these numbers will probably change since the publication of this document.

6.1 Communications

6.1.1 Connection

In web connections, Mega uses TLS 1.2, with 256-bit AES encryption in CBC mode, and it uses RSA as a key negotiation algorithm.

As RSA is used to negotiate the symmetric key, this means that the key is generated by the client and then encrypted with the public key in the certificate presented by the server (sent out by Comodo, as shown in Graphic 15), which is an 2048 bit RSA. Therefore, only Mega can decrypt the TLS key generated by the client and Man in the Middle type attacks are avoided.

\(^{39}\) [https://mega.co.nz](https://mega.co.nz).

\(^{40}\) [https://mega.co.nz/#help](https://mega.co.nz/#help).
6.1.2 Registration

The data that must be given to fill out the registration for an account in this service is: name, surname, email address and password, besides accepting the service’s conditions. When submitting the form with this data, an anonymous user is created (pending the confirmation of the indicated email), with a master user key that is made up of 4 integers (32 bytes).

The master key is sent to Mega’s servers encrypted with AES using a key derived from the password introduced by the user. The JavaScript code that generates these values, and a request example, can be seen in Graphic 16.

```javascript
if (u_k) api_create_u_k();

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

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

req = { a: 'up',
    k: u_to_base64(encrypt_key(new sjcl.cipher.aes(ctx.passwordkey), u_k)),
    ts: base64_noncekey(all_to_sstr(asc) + all_to_sstr(encrypt_key(new sjcl.cipher.aes(u_k, asc))));

[
  {"a":"up","k":"QR8Kx0rhnPH8frIEmj3HHQ","ts":"EQnAV0k0Dgi81bnD0rX_QzSmpAgUjDnt8oDlkGHPcW4"}
]```
The registration request is replied to with an activation email, sent to the specified address. In Graphic 17 the function responsible for this delivery can be observed, along with an example of a response sent from a client computer and the associated email that is received by the user.

![Figure 17. Delivery of confirmation email in Mega, sendsignuplink function, and example of data sent to the client through the browser and via email.]

```javascript
function sendsignuplink(name, email, password, etc)
{
    var pu_aes = new Yoj物.get_secret_key_pass(password);
    var enc = { 
        c: base64urilencodencplaintext.enkrypt_hex_au, 
        n: base64urlencodencplaintext.n, 
        m: base64urlencodencplaintext.m
    }
    mpi_enc(ecn, etc);
}
```

"["a":":uc","c":":PIAHhJxqNqItHemshK9fFkvr6KJ3M9AB9A3w5YCHBN","e":":tmkWeE3h2cg","m":":vhydihyq3hird3fheuuy29gc"]"

After accessing the link sent via email, the user is asked to introduce the password that was indicated when requesting registration once again. If the password is introduced correctly, the email address is sent to the server and is encrypted with an AES key derived from the password, as shown in Graphic 18. This value (“uh” in the graphic), is subsequently used as an authentication token to authenticate the user.

![Figure 18. Delivery of the registered email address, encrypted with the AES password derived from the password.]

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

Finally, an RSA keypair is generated (of 2048 bits), sending the private key encrypted with the user’s AES master key, as depicted in Graphic 19.
To sum up, during the registration process, two keys are generated:

1. An AES key, made up of 128 bits, referred to in this study as the AES master key.
2. An RSA keypair, of 2048 bits.

The private RSA key is encrypted with an AES master key, which is also encrypted with a 128-bit AES key derived from the password introduced by the user. In this encrypted form, both the RSA pair and the AES master key are uploaded to the server.

6.1.3 Login

When the client logs in, as shown in Graphic 20, the email address indicated by the user with the AES key derived from the user password is encrypted again, just like during the registration (see Graphic 18). If the correct address and password are introduced, the «uh» value generated is correct, and the service searches the user’s data and returns the user’s encrypted keys.
After this, the user’s browser does the following:

1. Decrypt the AES master key by using the password introduced by the user.
2. Decrypt the private RSA key by using the AES master key.

In Graphic 21 a fragment of the code in which these operations are executed can be seen.

Finally, it is important to point out that the platform does not support a double factor security in cloud storage services.
6.2 Information management

6.2.1 Transmission

As demonstrated in Graphic 22, Mega uses AES in CCM mode for file transmission. CCM is a block cipher mode defined in RFC 3610\(^1\), which combines the CTR (Counter) mode with the CBC-MAC mode, providing authenticity besides confidentiality.

For each file uploaded to the service, a new random AES key is generated. When uploading the file, this key is encrypted with the user’s AES master key and is also uploaded to Mega, as depicted in Graphic 23, where the key used to encrypt the file is sent in the «k» field. Therefore, the user may recover the file from anywhere (it must be recalled that the user’s AES master key is also stored in the server, encrypted by using the personal password).

---

Figure 22. Fragment of the encrypter.js file to encrypt files uploaded to Mega.

```javascript
for ( var i = 0; i < data.length; i += 0x10000 )
{
  // put data chunk into the heap
  var j = ( i + 0x10000 < data.length ) ? i + 0x10000 : data.length;
  heap.set( data.subarray(i,j)), 0x1000 );

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

  // decrypt data
  mac.ctr_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,(ctr/0x1000000000) >>> 0,ctr >>> 0);

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

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

\(^1\) http://tools.ietf.org/html/rfc3610
Figure 23. Fragment of the `ul_finalize` function for the delivery of the files encryption key and submission example.

```javascript
var req = { a: 'p', 
  t: dic, 
  n: {
    h: file.response, 
    t: 0, 
    m: ab_to_base64(ena[i]), 
    k: a325_to_base64(encrypt_key(a_h_aes, file.filekey)) 
  }, 
  i: requests 
};
```

Regarding **deduplication**, Mega explains in its privacy and security declaration (see the corresponding fragment in Graphic 24), that it reserves the right to delete a file if it proves that it is an exact duplicate of another file that already exists in another location of its servers.

Figure 24. Fragment of Mega’s service terms regarding deduplication.

Despite the fact that this aspect does not shed any light on their deduplication policy, it is an indication that this will happen at some stage. According to [Mega’s own blog](https://mega.co.nz/#blog_3), a deduplication takes place once the data has been encrypted. However, given that the blog entry mentioned is quite old (January of 2013), and that there does not seem to be more specific and contrasted information in other accessible resources, various verifications have been carried out to confirm how does Mega carry out management of duplicated files.

On one hand, the function responsible for uploading the files, located in the `upload2_21.js` file, executes the fragment of code shown on Graphic 25. The `fingerprint` function (located in the `crypto_49.js` file) is invoked during this process and the CRC of the file (without encryption) to be uploaded is calculated, linking the last modified date (`lastModifiedDate`) of the file and coding the result in Base64, as can be seen in Graphic 26. Then, the `Finish` function invokes the `callback` function which corresponds to the anonymous function passed as second parameter to `fingerprint`. In

---

42 [https://mega.co.nz/#blog_3](https://mega.co.nz/#blog_3)
this callback function it is verified whether there is a file with the same “hash” (which in this case is the CRC linked to the modified date) amongst the user’s own files. If it already exists, the \texttt{ul\_deduplicate} function is executed, sending the data shown in Graphic 27, where “n.h” is the calculated hash.

As a result of all of this, it can be deduced that \textbf{Mega carries out client side single-user deduplications at a file level}. In other words, if a duplicated file is detected in the user’s own account, it avoids the delivery of the file, applying the indicated mechanism. In any case, it must be pointed out that besides carrying out a client side deduplication of the unencrypted data its not, it is not a risk initially for the user’s privacy amongst the user’s own files.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fragment_code_single_user_deduplication.png}
\caption{Fragment of the code for single-user deduplication.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fingerprint_calculation.png}
\caption{Final part of the \textit{fingerprint} calculation for files to be uploaded.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{data_sent_single_user_duplication.png}
\caption{Data sent in case of a single-user duplication and example of generated values.}
\end{figure}

\begin{verbatim}
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_stat(self);
    });
} catch (c) {
}
\end{verbatim}

After carrying out this verification, if the service does not detect duplications within the files of the user, it proceeds to send it. On the other hand, in the present study it has been verified that \textbf{Mega also carries out server side cross-user deduplications of the files once they’re encrypted}, as they state in their blog.
After sending the file, meta-information regarding the file is sent to the server. Specifically, the data shown in Graphic 28 is sent, corresponding to a fragment of the `ul_finalize` function, located in the `upload2_24.js` file. The important data in this aspect is that associated to the “a” and “k” fields, the latter being the AES key used to encrypt the file, which is also encrypted with the user’s master key. The former is the name of the file and its hash, and both of which are encrypted with the key used to encrypt the file. This hash is the same one that is used in the single-user deduplication, in other words, it is a CRC32 that calculated from the data in the plain file amongst other elements.

In this case, it must be noted that the `enc_attr` function encrypts this data with the file’s key before sending it.

Figure 28. Fragment of the `ul_finalize` function, with the meta-information about the uploaded file and involved in the cross-user deduplication process.

```js
var body = { n: file.name }
if (file.hash) body.c = file.hash
var ea = enc_attr(body)
var fuid = file.fuid ? api_getfinfo(file.fuid) : false
var req = {
  a: 'p',
  t: dir,
  n: [{
    b: file.response,
    t: 0,
    a: sb_to_base64(es[0]),
    k: aes2_to_base64(encrypt_key(u_k_aes, file.filekey))
  }],
  i: request
};
```

It can be verified that the server-side cross-user deduplication is carried out by manipulating the aforementioned “a” and “k” fields. To do so, two “prepared” accounts (Account A and Account B) must exist. It is important to remember that in Mega there are three kinds of keys besides each user’s personal password: the AES master key; the AES keys for encrypting files, which at the same time are encrypted with the user’s master password; and the RSA keys for the sharing of files (this will be seen further on).

- In the first place, both Account A and Account B must have the same AES master key: this can be achieved, for example, by debugging the web application during the registration when creating the second account and modifying the appropriate values.
- In second place, two different files are necessary: a File A to be uploaded to Account A, and a File B to be uploaded to Account B. It is then assumed that File A is uploaded to Account A first, and that subsequently File B is uploaded to Account B.
• In third place, it is necessary to generate the same AES key for both files. Once again, this can be achieved by debugging the web application when uploading File B and modifying the appropriate values.

• Finally, when uploading the file, the value sent in field “a” to File B must be modified, establishing the same value that was sent for File A. Once this is done, the file that is downloaded from Account B will be File A, instead of File B. This process is schematized in Graphic 29.

**Figure 29. Scheme of the concept test to check the cross-user deduplication on Mega.**

Various points need to be made regarding this point:
• The need to make the keys coincide is only so that Account B correctly shows File A. If it is not changed, the deduplication also takes place (it seems that deduplication is based only on the value transmitted in the “a” field). However, when the JavaScript code downloads the list of files in the Account B, it detects that the meta-information cannot be decrypted correctly (given that the keys do not match), so the file is not shown despite the information having been sent to the client.

• The cross-user deduplication is carried out based on the result of the following operations (here they are simplified, for readability):

\[ a = AES(\text{nombre fichero} + \text{CRC32(fichero en plano)}, \text{clave fichero}) \]

In other words, if two files produce the same “a” value, the platform assumes that the last one to be uploaded is the same as the first one. Being “a” the result of an AES encryption with random keys newly generated for each file, the probability of a collision is quite remote.

• Besides, the delivery of the encrypted file is carried out regardless of whether it already exists in its servers. In fact, the “a” value on which the deduplication is based on is sent out after the delivery of the encrypted file. In other words, Mega does cross-user deduplication in terms of storage, but not in terms of the communication of data, which implies that it is not possible to carry out an analysis like the one shown in [7].

Finally, it can be confirmed that Mega cannot verify the hashes (CRC32) received from the files via the procedure explained previously to reproduce the deduplication. This is because, as it is based on the “a” value in the plain data, it cannot reproduce the calculations from the encrypted files it receives to verify if they have been executed properly.

6.2.2 Storage

As shown previously, the files are encrypted on the client’s side, using 128-bit AES that are generated randomly. These keys are also encrypted with the user’s AES master key, which is encrypted again with an AES key derived from the user’s password. Therefore, for Mega, it can be verified that files are stored, at least, with these guarantees.

6.2.3 Sharing

In Mega files and folders can be shared through links. In this case there are two options, shown in Graphic 30.
• Include the encryption key that gives access to the information in the link. Anyone who accesses this link will be able to directly download the file or access the content in the folder.
• Omit the encryption key, which is required to access the link.

**Figure 30. Sharing of files through links (with or without a password).**

Alternatively, the file or folder can be sent directly to a user. In this case, the AES key used to encrypt the file is encrypted using the public key of the recipient by calling the `encryptto` function, shown in Graphic 31, from the `this.copyNodes` function.

**Figure 31. Encryptto function.**

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

  if (pubkey = u.pubkeys[user]) {
    return crypto_rsaencrypt(data, pubkey);
  }

  return false;
}
```

Of the three possibilities (sharing the link with the key, sharing the link without the key, and directly sharing it with a contact), it is important to be cautious with the option that includes the key within the link, given that anyone who gets their hands on it (legitimately or illegitimately), can directly access the related information.
6.2.4 Deletion

When the user deletes a file, it is moved to the recycle bin. However, Mega does not specify in the terms of its service what it does with a file once it is deleted from the bin.

6.3 Other aspects

One of the main cryptographic functions executed on the client side via the cryptographic libraries downloaded from Mega’s servers is how it guarantees the integrity of these libraries. The process is divided in two phases:

1. By connecting with https://mega.co.nz, a first request is made to secureboot.js, as shown in Graphic 32, located in the same domain. As mentioned previously, the connection with https://mega.co.nz is protected with SSL, using 256-bit AES for encryption.

2. The `secureboot.js` file contains an array of all the JavaScript files that must be downloaded next, and that stores its hashes (SHA256). For example, the `crypto_48.js` file’s hash, from the 16/09/2014, shown in Graphic 33.

---

<table>
<thead>
<tr>
<th>Req. Timestamp</th>
<th>Método</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 16/09/14 11:06:41</td>
<td>GET</td>
<td><a href="https://mega.co.nz">https://mega.co.nz</a></td>
</tr>
<tr>
<td>3 16/09/14 11:06:41</td>
<td>GET</td>
<td><a href="https://mega.co.nz/secureboot.js?m=65694116">https://mega.co.nz/secureboot.js?m=65694116</a></td>
</tr>
<tr>
<td>9 16/09/14 11:06:42</td>
<td>GET</td>
<td><a href="https://eu.static.mega.co.nzlong/on_22.png">https://eu.static.mega.co.nzlong/on_22.png</a></td>
</tr>
<tr>
<td>11 16/09/14 11:06:43</td>
<td>GET</td>
<td><a href="https://eu.static.mega.co.nz/js/dist%5C_j%5C_1.js">https://eu.static.mega.co.nz/js/dist\_j\_1.js</a></td>
</tr>
<tr>
<td>12 16/09/14 11:06:43</td>
<td>GET</td>
<td><a href="https://eu.static.mega.co.nz/js/livestore%5C_3.js">https://eu.static.mega.co.nz/js/livestore\_3.js</a></td>
</tr>
<tr>
<td>14 16/09/14 11:06:43</td>
<td>GET</td>
<td><a href="https://eu.static.mega.co.nz/js/crypto%5C_48.js">https://eu.static.mega.co.nz/js/crypto\_48.js</a></td>
</tr>
</tbody>
</table>

---

The hash is stored as 8 integers of 4 bytes each, in other words: 4 integers x 8 bytes per integer x 8 bits per byte = 256 bits.
3. Next, the JavaScript files are downloaded, as can be seen (in part) in Graphic 32. In this case, the requests are made to https://*.static.mega.co.nz, using the same type of protection than for the connection with https://mega.co.nz.44
4. Once the files are downloaded, their hashes are verified with the contents in secureboot.js, sending back an error message if they do not coincide, as shown in Graphic 34.

Figure 34. Comparison of JavaScript file hashes.

```javascript
if (imcorrectcheck & & (fouerror||md5sum(md5sum(tljs[1].text),sll[1][tljs[1].f])))
{
    alert('An error occurred while loading MGDA. The file '+bootstaticpath+jsl[tljs[1].f]+' is corrupt.'
    contenterror=1;
}
```

Therefore, the integrity of the JavaScript files downloaded when logging in, which are responsible for the protection of all the information that is stored in its cloud, is guaranteed at the level offered by RSA (2048 bits) and AES (256 bits).

6.3.1 Personal management of passwords

Mega does not include support to importing or exporting the user’s keys generated from external tools.

6.3.2 Change of password

Keeping in mind the infrastructure that the service uses for cryptographic keys, when changing the password it is necessary to update the encrypted version of the AES master key stored in its servers, as it is stored in encrypted form with the user’s password. Otherwise, its decryption would give an error and the files which were previously stored would not be able to be recovered. Therefore, when a user changes his password, the AES master key is reencrypted using (see Graphic 35) the new password, and this new encrypted version is sent once again to the servers to update the previous one.

44Initially, as stated on https://mega.co.nz/#blog_3, a 1024-bit certificate was used, which generated some controversy.
Figure 35. Function to change a password on Mega.

```javascript
function changePw(currentPw, newPw,.ctx) {

    var pe_aes = new a2l.cipher.aes(prepare_key_pv(newPw));

    api_req({
        method: 'POST',
        url: 'http://localhost:3000/changePw?ctx=' + ctx + '&currentPw=' + currentPw + '&newPw=' + newPw,
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            newPw: newPw
        })
    });
}
```

### 6.3.3 Criticism of JavaScript-based cryptography

One of the greatest criticisms that Mega has received is the fact that it provides its functionality mainly through JavaScript cryptographic libraries, excluding mobile clients and Sync, for which Mega is a normal application that works on a web browser. For Firefox and Chrome applications, the JavaScript code is downloaded as part of a plugin for these browsers, meaning that it is not necessary to download it every time the web client is accessed.

Keeping the alternatives supported for Mega to distribute the code to one side, the use of JavaScript to implement cryptographic functionality is controversial. A number of experts in this field specifically oppose this practice, arguing the following aspects:

- The distribution of the JavaScript code is a weak point.
- JavaScript does not allow the generation of cryptographically secure random numbers.
- Immaturity of the existing cryptographic libraries.
- Malleability of the execution environment.

As for the first point, the mechanism to distribute JavaScript code has already been described. If plugins are not used, it is necessary to trust the legitimacy of the code distributed by Mega and, as it is transmitted every time a connection is established (unless if it has been cached), the time where it is exposed to attacks increases.

In any case, a mechanism is established for managing the download of JavaScript so that the transmitted code is always protected by TLS. Therefore, as long as the service distributing the code (or it is audited every time it is downloaded, which is practically impossible) and the certification authority issuing Mega’s certificate are trusted, this problem seems to be satisfactorily solved.

On the other hand, if plugins are used, trust must be placed both on Mega, as it is still responsible for distributing the code in the first place, and Chrome or Firefox, as they manage the plugins.

The problems regarding the generation of random numbers are undoubtedly important, as practically any cryptographic primitive depends at some stage on the robust generation of random numbers.

In Mega, this is resolved by using mouse movements or keystrokes as a source of entropy to renew the seeds that are used in random numbers generation. Graphic 36 shows a fragment of the `mouseMoveEntropy` function, in the mouse_6.js file, where the coordinates of the cursor are used.

Figure 36. Fragment of the `mouseMoveEntropy` function for the re-establishment of the seed for the generation of random numbers based on the event of the mouse and keyboard.

```javascript
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 = 100 first move events should provide at about 270 bits of entropy
    // (conservative estimation is 1.5 bits of entropy per move event)
    amsCrypto.random.seed( new Uint32Array( [ v ] ) );
  }
}
```

As for the third criticism, the cryptographic functionality provided is based on the Stanford JavaScript Crypto Library (SJCL 46), implemented in 2009 by experts from Stanford University. This library is probably the best valued cryptographic library in JavaScript today and since its creation it has been subject to an active maintenance process (see graphic of contributions shown in Graphic 37, available on GitHub).

Figure 37. Commits history on SJCL.

46 http://bitwiseshiftleft.github.io/sjcl/
Obviously, the code distributed by Mega may have suffered modifications despite being based on SJCL. The differences between part of one encryption and another can be seen in Graphic 38.

The comparison has been made once the syntax styles have been unified, given that Mega’s library contains more idioms than the official library, probably to minimize its size. As for the encryption, practically all the differences correspond to variable naming changes, with changes also existing in the code organization.

**Figure 38. Differences between the official SJCL and the one included on Mega.**

![Differences between the official SJCL and the one included on Mega.](image)

Finally, the malleability of the execution surroundings in JavaScript is surely the biggest problem to overcome, as it is inherent to the technological context. An example of this problem is the megaPWN bookmarklet, a browser bookmark that contains a small portion of JavaScript code.

Although this software does not seem to work in the latest versions of the most common browsers, when it was published it enabled access to unencrypted private passwords managed by Mega (see Graphic 39), according to its author. In this aspect, the only option is to trust in the code provided by Mega (or audit it every time it is modified) and minimize the risk that components external to Mega (which the user would have to install, either consciously or unconsciously) somehow modify its behaviour or infringe the integrity of the distributed code.

The good news in this aspect is that all modern browsers support the same-origin policy.

---


48 [http://nzkoz.github.io/MegaPWN/](http://nzkoz.github.io/MegaPWN/)

policy\textsuperscript{50}, the main policy to assure that a script from a certain origin can modify the properties loaded from that other origin. However, there have been cases where this mechanism could be circumvented as a result of weaknesses in its implementation\textsuperscript{51}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{megapwn.png}
\caption{Example of megaPWN execution extracted from its website.}
\end{figure}

6.4 Summary

- **Connection**: TLS 1.2, with 256-bit AES in CBC mode with RSA for the password negotiation.
- **Registration**:
  - The user’s email is encrypted with an AES key derived from the password introduced by the user as an authenticating token.
  - An AES master key is created, which is encrypted with the user’s password, along with an RSA keypair that is encrypted with the master key. Both keys are sent encrypted to the server.
- **Authentication and login**:
  - The authenticating token, which is derived initially during registration, is regenerated from the email address and password introduced by the user.
  - It does not support double factor authentication.
- **File transmission**:
  - Client side *single-user* deduplication at a file level.
  - Server side *cross-user* deduplication regarding encrypted data at a file level.
- **File storage**: Encrypted using Mega’s keys infrastructure.
- **File sharing**:
  - Via links in which the encryption key can be included or not.
  - Via contacts.

\textsuperscript{50} https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy
\textsuperscript{51} http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html
• **File deletion:** Not specified.
• **Other aspects:**
  o The cryptographic operations are carried out from the client side (JavaScript).
  o **Compatibility with a self-management of passwords:** No.
# Summary table

The following table is a summary of the main properties analysed in the previous sections. For specific details regarding any of them, consult the corresponding section.

<table>
<thead>
<tr>
<th></th>
<th>Dropbox</th>
<th>Mega</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Communications</strong></td>
<td><strong>Connection</strong></td>
<td><strong>Registration</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Registration</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Login</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Information</strong></td>
<td><strong>Transmission</strong></td>
<td><strong>Storage</strong></td>
</tr>
<tr>
<td><strong>Transmission</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Sharing</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
<tr>
<td><strong>Deletion</strong></td>
<td><strong>Dropbox</strong></td>
<td><strong>Mega</strong></td>
</tr>
</tbody>
</table>

## Communications

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td><strong>Connection</strong></td>
</tr>
<tr>
<td><strong>Dropbox</strong></td>
<td>TLS 1.2</td>
</tr>
<tr>
<td></td>
<td>AES-GCM 128bits</td>
</tr>
<tr>
<td></td>
<td>ECDHE and RSA</td>
</tr>
<tr>
<td><strong>Mega</strong></td>
<td>TLS 1.2</td>
</tr>
<tr>
<td></td>
<td>AES-CBC 256 bits</td>
</tr>
<tr>
<td></td>
<td>RSA</td>
</tr>
</tbody>
</table>

## Registration

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Password sent through TLS, without processing</strong></td>
<td><strong>Authenticating token generation</strong></td>
</tr>
<tr>
<td></td>
<td>Creates an AES master password encrypted with password</td>
</tr>
<tr>
<td></td>
<td>Creates RSA passwords encrypted with AES master password</td>
</tr>
</tbody>
</table>

## Login

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Password sent through TLS, without processing</strong></td>
<td><strong>Authenticating token regeneration</strong></td>
</tr>
<tr>
<td></td>
<td>Double factor supported</td>
</tr>
</tbody>
</table>

## Information

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client side single-user deduplication</strong></td>
<td><strong>Client side single-user deduplication</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Server side cross-user deduplication on encrypted data</strong></td>
</tr>
</tbody>
</table>

## Storage

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encrypted with AES-256 password controlled by Dropbox</strong></td>
<td><strong>Encrypted with AES-128 password controlled by client</strong></td>
</tr>
</tbody>
</table>

## Sharing

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Links not protected</strong></td>
<td><strong>Links (protected or not protected)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Delivery to contacts</strong></td>
</tr>
</tbody>
</table>

## Deletion

<table>
<thead>
<tr>
<th><strong>Dropbox</strong></th>
<th><strong>Mega</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In the server after 30 days</strong></td>
<td><strong>Not specified</strong></td>
</tr>
</tbody>
</table>
8 Bibliography


9 Links


2. [https://mega.co.nz](https://mega.co.nz). Last accessed on 01/25/2015.


10 Graphics table

Figure 1. Use of cloud storage systems in European Union countries. ...................... 7
Figure 2. Market share of cloud storage applications in the USA. Study carried out in 2013 by Strategy Analytics. ................................................................. 8
Figure 3. Server side deduplication. ................................................................. 11
Figure 4. Client side deduplication................................................................. 11
Figure 5. Single-user deduplication. ................................................................. 12
Figure 6. Cross-user deduplication................................................................. 12
Figure 7. Components to analyse a cloud storage application. ......................... 17
Figure 8. SSL connection with Dropbox......................................................... 19
Figure 9. Traffic capture during a Dropbox registration.................................... 20
Figure 10. Number of bytes transmitted from uploading the rnd.100 file to the A account. .................................................................................................................. 22
Figure 11. Amount of bytes transmitted by uploading the rnd.100 file to the B account. ................................................................................................................................ 23
Figure 12. Delivery of rnd.100.2 to the A account .......................................................................................................................... 23
Figure 13. Re-delivery of the rnd.100.2 file to the A account. ......................... 24
Figure 14. Deleted files in the Dropbox desktop client. ................................................. 25
Figure 15. TLS connection with Mega. ....................................................................... 28
Figure 16. Fragment of the api_createuser function in Mega and examples of generated values. .................................................................................................... 28
Figure 17. Delivery of confirmation email in Mega, sendsignuplink function, and example of data sent to the client through the browser and via email. .................. 29
Figure 18. Delivery of the registered email address, encrypted with the AES password derived from the password. .......................................................... 29
Figure 19. Delivery of the couple of RSA passwords, with a private password encrypted with AES and an example of generated values. .................................................. 30
Figure 20. Calculation of the authentication token when logging in. ...................... 31
Figure 21. Fragment of the api_getsid2 function where the user’s passwords are decrypted during login. .......................................................... 31
Figure 22. Fragment of the encrypter.js file to encrypt files uploaded to Mega. ...... 32
Figure 23. Fragment of the ul_finalize function for the delivery of the files encryption key and submission example. .......................................................... 33
Figure 24. Fragment of Mega’s service terms regarding deduplication. .................. 33
Figure 25. Fragment of the code for single-user deduplication. .............................. 34
Figure 26. Final part of the fingerprint calculation for files to be uploaded. ............ 34
Figure 27. Data sent in case of a single-user duplication and example of generated values ..................................................................................................................... 34
Figure 28. Fragment of the ul_finalize function, with the meta-information about the uploaded file and involved in the cross-user deduplication process. ............... 35
Figure 29. Scheme of the concept test to check the cross-user deduplication on Mega. ......................................................................................................................... 36
Figure 30. Sharing of files through links (with or without a password) .................. 38
Figure 31. Encryptto function. .............................................................................. 38
Figure 32. JavaScript file requests when loading Mega. ........................................ 39
Figure 33. crypto_48.js file’s hash on the 16/09/2014 ............................................ 39
Figure 34. Comparison of JavaScript file hashes .................................................. 40
Figure 35. Function to change a password on Mega. .............................................. 41
Figure 36. Fragment of the `mouseMoveEntropy` function for the re-establishment of the seed for the generation of random numbers based on the event of the mouse and keyboard. ................................................................. 42
Figure 37. `Commits` history on SJCL. .......................................................... 42
Figure 38. Differences between the official SJCL and the one included on Mega.... 43
Figure 39. Example of megaPWN execution extracted from its website. ............. 44
Figure 40. Convergent encryption in the context of cloud storage. ....................... 52
ANNEX 1 – CONVERGENT ENCRYPTION

Convergent encryption algorithms [8] are used to always produce a same encrypted text from a common plain text, so that only people who know the original plain file can recover it. Although this implies lower security in terms of the information privacy, since it does not guarantee the indistinguishability of the encrypted texts, it is a useful primitive to increase the efficiency of public storage services (or large scale private ones). To do so, they combine cryptographic hashes with symmetric and asymmetric encryption algorithms. In Graphic 40 the process of a defined convergent encryption [8] can be observed. The data is encrypted through a symmetric encryption algorithm, using a hash from the data itself as key. This key is also encrypted with a public key from the user, and the pairing formed by the encrypted data and the encrypted key is stored in the server.

When new data is added to the server, if it already exists only the key encryption process will be repeated, but in this case using the public key from the new user. Only the new encrypted key is uploaded to the server, which is linked in some way to the encrypted data (for example by updating a list of associated keys). This way, when a user wants to access the information, he will have to decrypt the adequate symmetric key using his public key. Therefore, only the users who have executed the previous process can obtain the correct symmetric key and finally decrypt the data.

However, it must be pointed out that the clients of the cloud storage applications do not always possess pairs of public-private keys, so convergent encryption schemes in these cases can vary from what has been explained before. For example, the list of symmetric encrypted keys (each one with a public key from a different user) could be substituted by an encrypted list of the users who possess the associated file with a key under the unique control of the service provider.

Figure 40. Convergent encryption in the context of cloud storage.