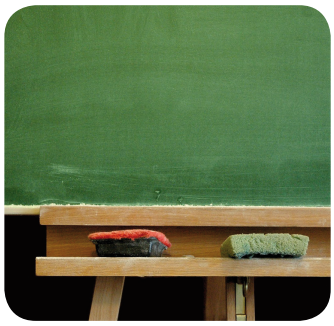
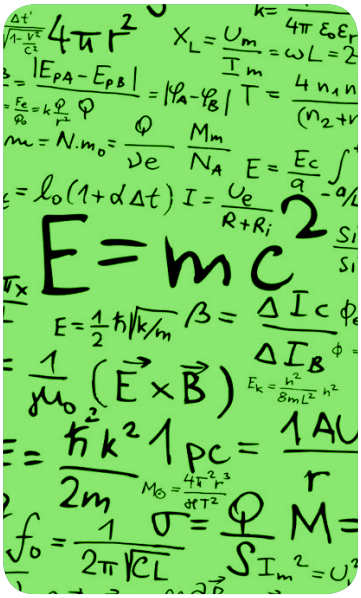
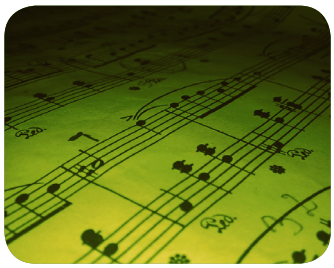






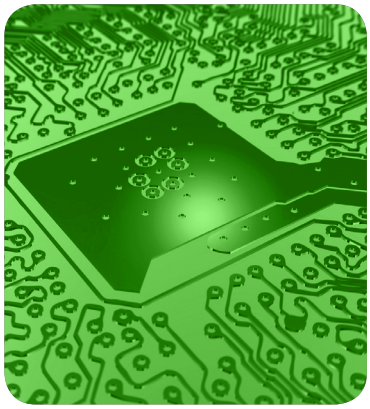
MÁSTERES de la UAM

Escuela Politécnica Superior / 15-16

Ingeniería Informática

UNIVERSIDAD AUTONOMA DE MADRID

excelencia Campus Internacional UAM CSIC+



Development of a software infrastructure for the secure distribution of documents using free cloud storage

Alejandro Sánchez Gómez

UAM EDICIONES

UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Development of a software infrastructure for the secure distribution of documents using free cloud storage

Máster en Ingeniería Informática

Autor: Sánchez Gómez, Alejandro
Tutor: Arroyo Guardeño, David

Madrid June 2016

Agradecimientos

Me gustaría recordar en estas líneas a todas aquellas personas que me han ayudado a conseguir mis objetivos y a finalizar este proyecto.

A mi tutor David, por la gran ayuda tanto en este proyecto como en el Trabajo de Fin de Grado, y por aconsejarme siempre que lo he necesitado.

A todos los profesores que he tenido durante los cinco años de universidad, siendo siempre muy cercanos al estudiante y preocupándose por su aprendizaje.

A todos mis amigos de la infancia, que siempre están conmigo cuando les necesito.

A las nuevas amistades hechas en la universidad, mención especial a Carlos, Borja, Iván, Laura y Rus. También quiero recordar a Javi y a Pascu, dos personas increíbles que he conocido en este máster.

A todas las nuevas amistades que hice en Bratislava gracias al programa Erasmus Prácticas, siendo muy importantes Raquel, Bea, Emma, David y Vratislav.

Y por supuesto, a mi familia, que es lo más importante y gracias a su esfuerzo he podido disfrutar de estos seis años de universidad.

A todos ellos, muchas gracias.

Alejandro Sánchez Gómez

RESUMEN

El siglo XXI pertenece al mundo de la computación, especialmente como resultado de la computación en la nube. Esta tecnología posibilita la gestión de información de modo ubicuo, por lo que las personas pueden acceder a sus datos desde cualquier sitio y en cualquier momento. En este panorama, la emergencia del almacenamiento en la nube ha tenido un rol muy importante durante los últimos cinco años. Actualmente, varios servicios gratuitos de almacenamiento en la nube hacen posible que los usarios tengan un backup sin coste de sus activos, pudiendo gestionarlos y compartirlos, representando una oportunidad muy económica para pequeñas y medianas empresas. Sin embargo, la adopción del almacenamiento en la nube involucra la externalización de datos, por lo que un usuario no tiene la garantía sobre la forma en la que sus datos serán procesados y protegidos. Por tanto, parece necesario el dotar al almacenamiento en la nube pública de una serie de medidas para proteger la confidencialidad y la privacidad de los usuarios, asegurar la integridad de los datos y garantizar un backup adecuado de los activos de información. Por esta razón, se propone en este trabajo Encrypted Cloud, una aplicación de escritorio funcional en Windows y en Ubuntu, que gestiona de forma transparente para el usuario una cantidad variable de directorios locales donde los usuarios pueden depositar sus ficheros de forma encriptada y balanceada. De hecho, se podría selecionar las carpetas locales creadas por la aplicación de escritorio de Dropbox o Google Drive como directorios locales para Encrypted Cloud, unificando el espacio de almacenamiento gratuito ofrecido por estos proveedores *cloud*. Además, Encrypted Cloud permite compartir ficheros encriptados con otros usuarios, usando para ello un protocolo propio de distribución de claves criptográficas simétricas. Destacar que, entre otras funcionalides, también dispone de un servicio que monitoriza aquellos ficheros que han sido eliminados o movidos por una tercera parte no autorizada.

Palabras Clave: Open Source, Cloud, Seguridad Cloud, Criptografía, Protocolos de Seguridad, Distribución de Claves, Integridad de Activos de Información.

ABSTRACT

The 21st century belongs to the world of computing, specially as a result of the so-called cloud computing. This technology enables ubiquitous information management and thus people can access all their data from any place and at any time. In this landscape, the emergence of cloud storage has had an important role in the last five years. Nowadays, several free public cloud storage services make it possible for users to have a free backup of their assets and to manage and share them, representing a low-cost opportunity for Small and Medium Companies (SMEs). However, the adoption of cloud storage involves data outsourcing, so a user does not have the guarantee about the way her data will be processed and protected. Therefore, it seems necessary to endow public cloud storage with a set of means to protect users' confidentiality and privacy, to assess data integrity and to guarantee a proper backup of information assets. For this reason, in this work it is proposed Encrypted Cloud, a desktop application which works on Windows and Ubuntu, and that manages transparently to the user a variable amount of local directories in which the users can place their files in an encrypted and balanced way. Therefore, the user could choose the local folders created by the Dropbox or Google Drive desktop application as local directories for Encrypted Cloud, unifying the free storage space offered by these cloud providers. In addition, Encrypted Cloud allows to share encrypted files with other users, using for this our own cryptographic key distribution protocol. Note that, among other functionalities, it also has a service that monitors those files which are deleted or moved by an unauthorised third party.

Keywords: Open Source, Cloud, Cloud Security, Cryptography, Security Protocols, Key Distribution, Information Asset integrity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Main objective

The development of Information and Communication Technologies (ICTs) is an essential component of the current economic model [1]. In this scenario, it is gaining an increasing interest the so-called cloud computing technology. Cloud computing enables the storing and processing of data without deploying a complex ICT architecture. It is well-known that the cloud computing paradigm is a growing trend in computing that every day is taking a leading role in people's lives. However, it should be noted that when users place their assets on a cloud server, there are no guarantees about the way the Service Provider (SP) manages them and thus the user cannot fully trust that server.

The use of low cost storage technologies is highly relevant in the context of Small and Medium Enterprises (SMEs) [2]. The development of cloud technologies, indeed, allows SMEs to have servers and store their resources for free or at low cost. That is why nowadays the migration to cloud environments is one of the most important concerns for organisations. Certainly, the recent financial situation makes each entity to look for opportunities to reduce costs without eroding the efficiency and quality of the provided service. In addition, these solutions should meet usability criteria to foster their easy adoption by end users or clients. Again, cloud computing offers a set of usable solutions, from the perspective of both end users and technology developers.

However, regarding the security of the information assets of a company, it cannot be ignored that cloud servers and platforms do not constitute a fully trusted third party. Therefore, before migrating to the cloud, users need to understand the implications of storing the information assets of their businesses on the infrastructure of a Cloud Provider (CP). In fact, the scenario drawn by the cloud introduces new security challenges and additional privacy issues [3]. Recent security problems in the

landscape of free-access cloud technologies lead us to confirm this matter. One of these incident which had a significant relevance took place in the last October of 2014, when around 700,000 Dropbox cloud server accounts were broken [4].

Furthermore, it is remarkable that the "cloud user" has limited resources to verify the integrity of her data (which is necessary to prevent unauthorised use of such information by the CP). Moreover, data integrity is a requirement to guarantee the consistency between the data and the business activity. Remember that a key feature of cloud computing is that the information of the users is stored in multiple geographically dispersed data centers. Thus, having multiple locations for files can be a threat towards this consistency, and that is why some mechanism of re-centralisation is required. Therefore, taking into account the dependence of the business model of our society on ICTs [5], it seems necessary to endow public cloud storage with a set of mechanisms in order to protect users' privacy and confidentiality and to asses data integrity.

For this reason, we present in this work Encrypted Cloud, a desktop application based on a prototype previously developed in the author's previous Final Bachelor's Project [6]. Encrypted Cloud provides the users of free cloud services with a control system that adds confidentiality and integrity to the availability supplied by these services. Thus, it could help users to continue making the most of all the advantages provided by these cloud technologies, but without paying in exchange for the privacy of their information. In addition, Encrypted Cloud allows the user to select a variable number of local directories in which the user wants to store her encrypted assets. In this way, the user could choose the local synchronisation directory of CPs like Dropbox or Google Drive, unifying the free storage space that they provide [7]. Moreover, Encrypted Cloud does not depend on CPs' APIs, which makes its maintenance easier. Finally, note that the proposed solution is designed in order to be improved by potential software developers: one of the main objectives is to make an easily extensible interface and it also has the importance of being an Open Source application. This aspect is really significant, because if a community of developers who are involved with the development of this free Open Source solution could be set up, the evolution of this software would be much faster and with higher quality and security [8].

1.2 Similar solutions

Nowadays, there are some cloud solutions which are more focused on trying to solve the security problems which threaten cloud storage environments. In fact, some

of them are really well-known, like Cloudfogger[1], Boxcryptor[2], Sookasa[3], EnCifra[4] and ScapePrivacy[5]. CipherCloud[6] is another platform that serves as a gateway for data encryption in real-time and before sending them to a cloud environment. They emphasise that the encryption keys are stored locally, so they are not shared with the CP. However, the main problem in all of these implementations is that they are private solutions. To begin with, the user has to sign up in a service which probably stores her authentication data. Therefore, these solutions have a server in which the user should not trust if she wants to have the control over her data. Moreover, in these types of services it is even more difficult to verify the security requirements than in Open Source projects, because in this last case the source code is available for the whole community.

Another well-known solution is Mega, which is a relatively new cloud storage option in the industry compared to the more established solutions such as Google Drive, OneDrive and Dropbox. Mega also has its own local clients for Windows, OS X and Linux, but its main features are the 50GB of free storage space that it provides, and its high level of security, since it offers client-side encryption of data which means files are encrypted before they are uploaded to the server. In fact, Mega claims that it does not have any way of accessing user's information, as Mega allows the user to hold her master decryption key. It means that anything the user stores in her account is only accessible by her. The main problem for the user comes in the context of key management, since if the user loses her master key, there is no way to recover her account, so her files will remain encrypted and stored on the server [9]. However, as it has remarked before, the user should not trust this CP, as we can conclude from the analysis of MEGA by the MEGApwn project [10]. In this study it is shown that the encryption master key in MEGA is not actually encrypted and can be retrieved by MEGA or anyone else with access to the user's computer.

Currently, there are two solutions which are really similar to Encrypted Cloud. The first one is OmniShare [11], which protects cloud users with client-side encryption with strong keys without significant reduction in the user experience. This solution is also available on different types of devices (mobiles, desktop). However, its key distribution protocol requires that these devices have to be close in order to make

[1] www.cloudfogger.com
[2] www.boxcryptor.com
[3] www.sookasa.com
[4] www.encifra.net
[5] www.scapeprivacy.com
[6] www.ciphercloud.com

the key exchange. The other solution is Cryptomator[7], a free client-side encryption for cloud files which does not have a key distribution protocol for sharing encrypted files: the user is in charge of distributing the encryption key.

1.3 Structure of this document

This document is structured as follows. We start in Chapter 2 with the history and the current state of the world of cloud storage. In addition, for the correct understanding of the following chapters, it is needed to remind that in the cloud there are two different entities. On the one hand, the cloud user is the main actor who uses cloud services to support its specific strategic goal and intention. On the other hand, the CP has two unique properties: service and deployment model to support the cloud users. Chapter 3 presents ten security and privacy challenges that could be found in free-access public cloud storage services. We discuss the main concern of each of these challenges, and we provide a set of recommendations to tackle them. These recommendations are intended to identify cryptographic procedures and software solutions that can help both SMEs and end users to implement usable and low cost security solutions for free-access cloud storage. In Chapter 4, Encrypted Cloud is explained, describing in detailed different aspects of this project like the planning, the design, the implementation, the functionality description and the test plan. Finally, we draw the main conclusions of this work in Chapter 5 along with several improvements over the project that could be the starting point of future work.

[7] https://cryptomator.org

Chapter 2

Cloud storage

2.1 History

The changes in data storage is one of the major characteristics of the evolution of information technologies [12]. It provides a clear example on the construction of abstract models to generate and manage information [13]. Certainly, in the beginning of modern communication networks, data value was very close to the physical nature of the data storage medium. In the 1950s, the access to ICTs was restricted to big companies and governmental organisations. As ICTs evolved, citizens were able to access and buy Personal Computers, which was concomitant with a popularisation of the Internet. In this new scenario, most citizens could manage information assets through local data storage media as hard drives, floppy disks, CD-ROM, DVD, and USB-drives. Moreover, citizens and companies were able to make their information accessible to third parties through private and/or public web servers. Internet hosting services, in fact, is one of the most important elements configuring the digital transformation of business and society. Data outsourcing, nowadays, is a common practice, and more and more citizens and companies use SPs to manage information assets. Consequently, either implicit or explicitly, users and companies trust third parties, i.e., the SPs, to handle information [14, 15].

On the other hand, the concretion of new interfaces to access data has an impact not only on the users of such interfaces. Indeed, the creation of new *modalities* to generate and control data has determined a paradigm change in software development. The increasing complexity of operating systems and the geographic distribution of resources have determined a *divide-and-conquer* methodology in software development. There is clear urge to decouple content management from data management, for the sake of agile development of new products and the efficient adaptation to changes in the production and/or exploitation environment [16].

2.2 The state of cloud storage

Cloud storage is a cloud computing model in which data are stored on remote servers accessed from the Internet. Therefore, it provides a whole set of solutions to ease user access to data, which supplies an organisation with tangible competitive advantages: significant costs savings, improved degree of scalability, flexibility and resource pooling availability [17]. In addition, this new paradigm allows the creation of innovative software products by developers.

However, in cloud environments data management is delegated to third parties according to different trust models [18, pp. 246-251], which represent concrete implementations of security requirements and expected features. Actually, in cloud storage we have to distinguish between private, community, public and hybrid trust models [19]. Privacy threats differ depending on the type of cloud scenario and threats such as lack of user control, potential unauthorised secondary usage, or data proliferation are more dominate in public cloud [17].

Moreover, from the perspective of software developers, we should consider a more general landscape and not restrict ourselves to cloud storage and ponder over cloud computing. In cloud computing, a developer can opt for different trust models according to the so-called Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), which constitute the SPI (Software, Platform, Infrastructure) model [20]. This trust ranges from a total control of cloud technologies (Infrastructure as a Service model), to a complete trust in the cloud SP (Software as a Service model). In Chapter 3 we analyse in detail this last trust model, underlying the main risks and solutions for both domestic users and SMEs working with free public cloud storage solutions.

Chapter 3

Main threats and challenges in cloud storage

3.1 Introduction

Over the last years, we have witnessed a lot of attacks and security holes in ICTs[1]. Big companies have been compromised by hackers, which has derived in the erosion of their reputation and subsequent economical losses. These cyber-attacks have been possible due to the existence of some vulnerability or misconfiguration of the traditional procedures to secure information systems. Certainly, the deploy of information security systems is a very complex task. In today's scenario this is even more elusive.

Nowadays, mobility is a major fingerprint of ICTs. The increasing in the use of mobiles, tables or VPNs, along with the huge quantity of data stored in the cloud, make really difficult to delimit a boundary with which a user could have a total control over her assets. As the boundaries of data storage are less defined, the same thing happens with the security perimeter [21]. When it is not clear the perimeter which we have to secure, we should consider how an organisation can protect itself against data leakage and malware. Therefore, it is clear that cloud customers never access the "real" network or hardware, since they work inside virtual constructs. As a result, the adequate evaluation of the security of cloud computing requires to define different models, different tools, and new foundations [22].

This new security perimeter, together with the huge rise in cloud adoption and the ever-decreasing cost of deploying applications and services in the cloud, have made that cyberattacks against cloud systems have increased in the last years [23]. In fact, security breaches constitute the main concern that companies have in mind before migrating their business to this environment (see Figure 3.1).

[1] See http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/ for a good summary of the most conspicuous cyber-attacks and their impact.

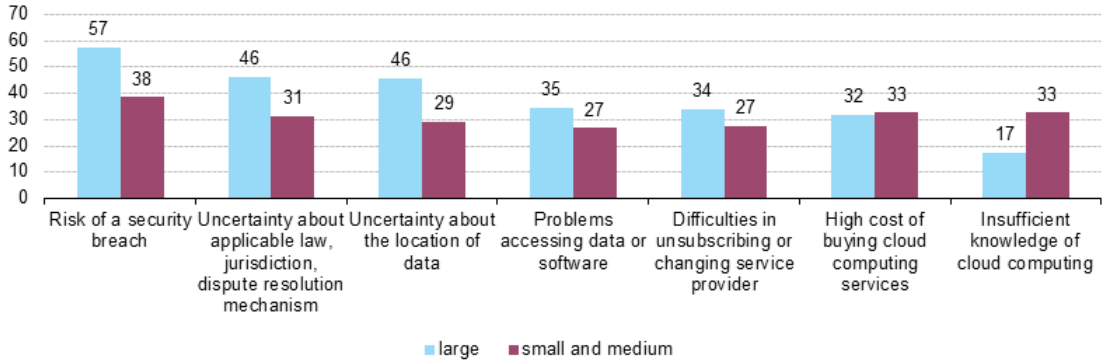


Figure 3.1: Factors limiting enterprises from using cloud computing services, by size class, EU28, 2014 (% enterprises using the cloud) [24].

These users' concerns, and this high amount of attacks whose target is cloud storage services, have motivated us to present in the following subsections ten security challenges which could compromise the user's security when she uses public cloud storage. Thus, the main goal of this chapter is to provide both companies and end users with a set of solutions in order to mitigate the well-known security issues in cloud environments.

3.2 Challenges

3.2.1 Challenge 1: Authentication

The protection of information assets demands the convenient concretion of access control methodologies. These methodologies are mainly based on authentication and authorisation procedures. The authentication process consists of verifying the identity of an user requesting access to a certain information asset. In the context of cloud services, this process is usually performed by checking users' credentials so it must be done in a safety way. However, CPs often use basic authentication mechanisms, in which the user has to provide her credentials to her provider. Therefore, this user has to trust the fact that providers do not have access to her credentials and they also manage these credentials in a secure way. Hold Security, which is an Information Security consulting firm, shows that users should not rely on CPs: this company has recently recovered 1.2 billion stolen credentials that belong to different companies [25], showing a poor management of the users' credentials [26].

During the last years, there have been several authentication problems in different cloud services. One of these is known as Man in the Cloud (MITC) attack [27], which has focused on file synchronisation services in the cloud, that allows individuals to

store synchronised copies of files repositories in multiple devices. This attack is based on the following: when a user of a synchronisation application wants to make an initial connection to an account or to switch between accounts, it requires that the user provides her credentials (username and password) in an interactive way through the user interface. Since this moment, the synchronisation application relies on a synchronisation token, which is obtained after this authentication process, for further operations. This token is stored in a register or in a file in the user's computer, whose location is fixed and depends on each provider and it is also independent of the machine. Therefore, by copying this token from an account that is controlled by an attacker in the correct place in the victim's account, the attacker will be able to change the synchronisation application to the account that is represented by her token. Then, the attacker can gain access to the accounts of the victims without compromising their passwords. Without using the user's credentials in a explicit way, this attack cannot be detected by the owner of the account. We want to remark that this security hole has its root cause in an insecure implementation of the OAuth 2.0 protocol, which is designed for authentication and authorisation between applications [28, 29].

Therefore, the user has to make an effort and has to be sure to use a strong authentication mechanism in the cloud. This measure could help her to avoid security breaches like for example the incident called "Celebgate", which was on the last August of 2014, where almost 500 private pictures of various celebrities were posted on public web pages, which were obtained from iCloud [30].

3.2.1.1 Solutions

In order to face this authentication problem, it is advisable to employ advanced password-based authentication techniques, like the Password-based Authenticated Key Exchange (PAKE) [31] or Secure Remote Password (SRP) protocols [32], which effectively conveys a zero-knowledge password proof from the user to the server. With this last mechanism, an eavesdropper or man in the middle cannot obtain enough information to be able to brute force guess a password without further interactions with the parties for each guess. This means that strong security can be obtained since the server does not store password-equivalent data.

Furthermore, this verification process of user's identity could be better by adding Two-Step Verification (2SV) mechanisms, which is present in several free cloud storage services [33]. In Google Drive, the user could use Google Authenticator [34], which is a two factor verification mechanism that adds an extra layer of security to users' accounts. Dropbox has also an optional 2SV (via text message or Time-Based One

Time Password apps) [35]. Box also provides the use of 2SV. In this case, the process requires a user to present two authenticating pieces of evidence when they log in: something they know (their Box password) and something they have (a code that is sent to their mobile device) [36]. Moreover, all OneDrive users can protect the login via a One Time Code app or a text message [35].

On the other hand, in some situations in cloud storage, users are required to register using Personally Identifiable Information (PII) at a CP to setup an account. Consequently, the provider can monitor the storage consumption behaviour of its users, i.e., every store, read, and delete operation can be tracked, and the provider thus has access to a complete behavioural profile of its users. Therefore, in this cloud services an anonymous authentication mechanisms are mandatory if the user really needs to get an adequate privacy in the cloud [37]. Another way to achieve this type of authentication is through non-conventional digital signatures, like group signatures, which allow members of a group of signers to issue signatures on behalf of the group that can be verified without telling which specific member issued it [38].

Finally, we would want to point out some possible solutions to the MITC attack that we have mentioned in Section 3.2.1. One possible strategy could be, firstly, to identify the compromise of a file synchronisation account, using a Cloud Access Security Broker (CASB) solution that monitors access and usage of cloud services by the users; and secondly, to identify the abuse of the internal data resource, deploying controls such as Dynamic Authorisation Management (DAM) [39] around the data resources identifying abnormal and abusive access to the data [27]. However, we consider that the best way to solve this security issue is being very carefully in the OAuth implementation. To achieve this goal, it is really advisable to be aware of the current potential weaknesses of this protocol, and to follow the well-known recommendations made by the cryptographic community about how to implement OAuth 2.0 in a secure way [28, 29, 40].

3.2.1.2 Limitations

In this and other scenarios, the deployment of security protocols imply a risk that has to be taken into account. As much effort used in the authentication process, we always can found a poor implementation of the cryptographic functionalities used, as it happened in the MITC attack, where an insecure implementation of OAuth 2.0 protocol comprised the security of several file synchronisation services. Heartbleed was also an important security hole [41], which was caused by an improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension, and it allowed to steal usernames and passwords from vulnerable sites.

Another bug which had a significant relevance in the last January of 2015 was Ghost [42], that was caused by a buffer overflow in a system library, called glibc, that is used in Linux distributions. This bug allowed attackers to remotely take complete control of the victim system without having any prior knowledge of system credentials.

Regarding 2SV mechanisms, it has been shown they are not completely secure, since there have already been several situations in which an attacker could "break" this mechanism [43, 44]. In fact, in December of 2012, the Eurograbber banking Trojan had a significant impact. This trojan was used to steal more than 36 million Euro from some 30,000 retail and corporate accounts in Europe [45]. This attack infected the computers and mobile devices of online banking customers. Consequently, 2SV mechanism was circumvented and the attackers were able to validate their illicit financial transfer [46]. Besides, if we are using a service from our mobile phone which includes 2SV sending a text message to this phone, the authentication process would be almost the same than if we only have to provide our username and password straightaway to this service [43].

On the other hand, we would like to add that nowadays it is widely used in several systems the so-called Single Sign-On (SSO) mechanism, which is a process that allows a user to enter a username and password in order to access multiple applications. Although it increases the usability of the whole system, these type of services, in which a token is used for authenticating a user, could be vulnerable to replay attacks [47]. As a result, the adequate combination of usability goals and security concerns is not easy to achieve, and the design of any procedure to improve usability should be thoroughly analysed before deploying the corresponding authentication process [48].

3.2.2 Challenge 2: Information encryption

Information encryption is an important aspect that users of Free Cloud Storage (FCS) should take into account. Certainly, the trust model and the risk model are very different when the SP is in charge of data encryption. If so, users are implicitly trusting SP, and this assumption can entail a threat to security and/or privacy [49].

With the current FCS solutions, users have to completely trust their CPs, since although these providers usually apply security measures to the service they offer, such measures allow them to have full access to the data which they store [50]. Indeed, even though the data are encrypted in the CPs servers, the user should pose herself the question about who has the control over the cryptographic keys used for this encryption. It is not ideal that this owner would be the CP, because in this case users would not have direct control on who can access their data.

Furthermore, providers could also be compromised because of the hardware failures, the flaws of software and the misbehaviours of system administrator. Once one of these happens, a tremendous amount of data might get corrupted or lost [51].

3.2.2.1 Solutions

Solutions for protecting privacy require encrypting data before releasing them to the CPs, and this measure has gained a great relevance since Snowden's leaks [52]. Client-side encryption could be properly articulated to obtain privacy and data integrity protection.

Major SPs as Microsoft, Yahoo and Google have led the way by adding End-to-End Encryption (E2EE) in the user data that they stored and managed. For example, Google Cloud Storage automatically encrypts all new data before they are written to disk [53].

On the other hand, encryption can bring several difficulties in scenarios where querying data is necessary. In these particular situations, we could use fragmentation instead of encryption, so we only have to maintain confidential the associations among data values in contrast to the values themselves. This technique protects sensitive associations by splitting the concerned pieces of information and storing them in separate un-linkable fragments [50]. Moreover, from the point of view of the CPs, fragmentation is a possible optimisation, since they could detect redundant file blocks which would be impossible with data encryption [54].

3.2.2.2 Limitations

The information security requires the protection of its confidentiality, i.e., it is only accessible by only those who have permission. As stated at the end of the last subsection, data fragmentation is a possible solution for achieving this confidentiality. However, its main limitation is that it must be managed by the user rather than by the provider.

Cryptographic is another alternative to guarantee this confidentiality. When the custody of the information corresponds to third parties, the user has to decide who is in charge of encrypting the information. On the one hand, if this encryption is managed by the provider, the user's privacy could be compromised, as it was shown in Snowden's leaks. Certainly, if the user wants to have complete control over her information assets, client-side encryption seems to be a good option. Nevertheless, it implies that the user is in charge of keys generation and management [53], which could suppose a high workload for her, along with a security risk (e.g., the user could

loss a cryptographic key and thus she could not access her data). Thus, it is clear that it would be necessary to have a mechanism in order to face this problem.

In this context, some applications allow user to generate and manage her keys using for this external tools [55], to subsequently import them into the client application of cloud storage. Therefore, as long as this tool also follows the security recommendations of key generation, this option is the strongest, since it allows user to choose tools that were specially designed for key generation and in which she trusts [56]. The first security recommendation is to use random number generators, to the possible extent, to have solutions based on One Time Passwords (OTPs), or at least, with limited password reuse. The next step is the use of key derivation functions like PBKDF2 [57], in which the OTP previously generated could be used to derive one or more secret keys. This two steps composed an ideal key generation process. However, some security breaches have been discovered recently in some key management tools, like in the case of LastPass [58]. For this reason, the user should not trust them and look for alternative solutions to avoid Single Point of Failure (SPOF) situations. Instead, the user could adopt the approach of distributed authentication, which consists of the generation of a strong secret from a password via interaction with two or more servers. With this new perspective, the user only has to know her password, and the compromise of one server exposes neither any private data nor any password [59]. Dyadic[2], a Multi-Party Computation (MPC) solution, meets with these requirements, since it protects secret keys and credentials by splitting them randomly between two or more servers. The unique property of Dyadic's technology is that all operations take place without ever bringing the key together, and so it is never in any one place to be stolen. Attackers need to simultaneously control multiple servers in order to learn anything, and this is made hard by using different operating systems, administrator credentials and more [60].

Apart from the key management problem, the user could lose some functionalities if she finally would decide to encrypt her data, since data encryption is not compatible with traditional deduplication mechanisms and she could not be able to make queries by keywords in the cloud [50]. However, Dyadic solves these problems working with homomorphic encryption (which we will present in Section 3.2.6). An alternative similar solution is Sharemind[3], which is also a MPC framework that can process encrypted data without decrypting it, using cryptographic methods which are costly (in terms of computing power) but efficient with the current technologies [61].

es

[2] www.dyadicsec.com

[3] http://sharemind-sdk.github.io/

Encryption of shared assets

Finally, the sharing of documents over the cloud is another feature that could be even more difficult to achieve when working with encrypted data. In this situation, it is necessary to share the encryption key of a encrypted file among the members of the group.

A first approach to solve this problem could be to make up a data packet based on the concept of the digital envelope container [62]. This packet will include the original document encrypted with a randomly generated symmetric key. Then, this key will be encrypted with each asymmetric public key which belongs to each member of the group and it will be attached to the packet. Therefore, when a member of the group who is not the owner of the document wants to open this document, first she would have to decrypt the symmetric encrypted key with her asymmetric private key, and then, she will be able to decrypt the document [6]. However, the main limitation of this approach is the time that the owner of the document needs to build up the whole packet, and also the time that a member of the group have to spend to get the encryption key of the document. Besides, when a user decides to join/leave the group, all the documents that this group is sharing have to be updated to grant/remove the permissions to this user.

Another approach could solve this key management problem through a centralised cloud service [63], so that encryption keys are stored by the CP, and it acts as a central hub of communication with users that ask for keys. The main problem is that it makes the user to trust completely the CP, because it generates all the private keys, so it has the ability to decrypt all the data and communications in its domain. Furthermore, this solution is affected by a SPOF threat. Certainly, if the keys of the CP are stolen, then all communications are compromised; if it suffers from a denial-of-service attack, then it cannot distribute private keys [63].

The third approach includes key management through a trusted client-side authority [63]: users are segmented into populations called groups, each of which has read and write access to a different data partition. Data partitions within the cloud are encrypted so as not to reveal information to the CP. A trusted intermediary, called manager, is in charge of a group of users of any size, and it is responsible for all aspects of its security: it generates and holds all private keys, allowing read and write access to the data partition. The main advantage of this solution is that the manager is under the control of the client organisation, and ensures that key management functions need not be outsourced to an untrusted CP. Additionally, each manager handles the authentication of only a limited set of users that interact with its own data partition, so it reduces a possible overload.

The fourth and last approach is more advanced than the previous ones [64], and it proposes a framework that combines together into a protocol a proxy signature, with which the group leader can effectively grant the privilege of group management to one or more chosen group members; an enhanced Tree-Based Group Diffie-Hellman (TGDH), that enables the group to negotiate and update the group key pairs with the help of cloud servers; and a proxy re-encryption, that allows to delegate most computationally intensive operations to cloud servers without disclosing any private information. Therefore, this solution supports better the updating of encryption keys, because it transfers most of the computational complexity and communication overhead to cloud servers without leaking the privacy.

3.2.3 Challenge 3: Inappropriate modifications of assets

In cloud storage the data owner loses the control over her assets, which means a change about how the enterprise perimeter is built up. From a classical point of view, this perimeter was constructed upon the assumption of a fixed geographic location for the underlying ICT infrastructure, visible for the owner of all IT assets. This allows to protect the assets using several security controls, as firewalls, Intrusion Detection System (IDS), etc., which collectively forms the security perimeter. However, in the cloud era, both location and data's owner are blurry and not clearly identified. Therefore, some of these security controls that are used in the old perimeter cannot protect IT assets in the cloud [21].

In this situation, it is not enough to have a well-defined security perimeter. In fact, since when the user uploads her assets to the cloud, she cannot be completely sure about when she later accesses her data again, it have not been modified by unauthorised third party, since an attacker can make small modifications in her assets that would be difficult to perceive. Even if the user has encrypted her data before uploading them to her CP, they can be modified and the user probably will not notice it.

On the other hand, for security considerations, previous public auditing schemes for shared cloud data hid the identities of group members. However, the unconstrained identity anonymity will lead to a new problem, that is, a group member can maliciously modify shared data without being identified. Since uncontrolled malicious modifications may wreck the usability of the shared data, identity traceability should also be retained in data sharing [51].

3.2.3.1 Solutions

The first solution that the user could have in mind is to perform a hash function in each file that she wants to upload to her CP, so then, when she accesses to these files, she could perform again this hash function over the downloaded file, and verify if it is the same as the first hash calculated. However, if this user has to deal with a big quantity of files, she could find the problem of where she can store all these hashes.

Another approach for solving this problem that the cryptographic community proposed was an online mechanism to verify both data availability and integrity, that consists of inspecting a small portion of these data combining a cryptographic processing. These methods are called retrievability proofs, which seems to have been first defined by Juels and Kaliski [65]. Their idea was to encrypt the information and then insert data blocks called "sentinels" on the encrypted information in random points. Sentinels are generated using a cryptographic function whose values cannot be distinguished from the encrypted data without knowing the function's parameters. In this way, a file is prepared to be sent to the cloud server to store it. The provider only sees a file with random bits, so she is not able to distinguish between what parts of data are original and what parts have been inserted randomly (sentinels). The data owner can verify if her data are intact, asking for a random selection of the sentinels that are returned by specifying the positions that only the owner knows. As the provider does not know where the sentinels are, any change in data can be detected [6]. However, this solution supposes an overload for the user who has only limited computing and network resources, because she would have to be in charge of verifying from time to time the integrity of her data stored in the cloud.

At this point, we should distinguish between content integrity (achieved with hashes, CRCs, sentinels, etc.) and both content and source integrity, which can be achieved with specific cryptographic methods. Therefore, we can agree that an exhaustive verification protocol is necessary in order to guarantee both data coherence and data integrity. As we have already seen, integrity verification is really important in all cloud computing schemes where the user does not trust provider's service [66]. Although it is possible to reach a conclusion about physical identity, it is not easy to establish a relationship between physical identity and digital identity. Cryptography provides the means for associating a digital identity to a user through asymmetric cryptography [38]. Consequently, the verification of the integrity of IT assets has been conceived and managed through conventional digital signatures.

However, in modern IT networks conventional digital signatures do not offer the whole of set of features that are required. In some situations, it is necessary to design a

signature scheme in which the message or document has to be signed by multiple users (for example, if we are dealing with a committee that must sign the whole document). On this ground, we have to take into account that there exist over 60 digital signature models [38], whose classification is not easy, as they have different characteristics. This being the case, group signatures could be used as anonymous authentication methods in the cloud. On the other hand, multi and aggregate signatures could be used when a file is digital signed and shared between several users in a cloud storage service, avoiding to produce multisignatures that are of linear size in the number of participants and with linear verification time (also depending on the participants). Finally, identity-based signatures could eliminate the need of distributing public keys in the cloud, allowing the verification of digital signatures just from the identity that the signer claims to own [38]. The transition from the need of communication networks and the features provided by these different schemes is not an easy task. In fact, this connection should be guided by strict evaluations of the theoretical bases.

Moreover, as we have seen at the end of Section 3.2.3, the auditing of shared cloud data demands the traceability of users' operations. However, this traceability could imply a risk in terms of users' privacy. Therefore, it is required an efficient public auditing solution that can preserve the users' privacy and, simultaneously, guarantee the identity traceability for group members. In [51] it is proposed an scheme in which a group manager is introduced to help members to generate authenticators and protect users' privacy. More specifically, identity traceability is achieved by two lists that record members' activity on each data block. Besides, the scheme also provides data privacy during authenticator generation by utilising a blind signature technique.

3.2.3.2 Limitations

Digital signatures are often performed using the asymmetric cryptographic algorithm RSA. The main disadvantage of this algorithm is the time required for its key generation, although it usually has to be done just once. Nevertheless, we could improve this time using elliptic curves [67], which provide the same security as RSA but with smaller key size. In fact, RSA key generation is significantly slower than ECC key generation for RSA key of sizes 1024 bits and greater [68, 69]. In addition, the cost of key generation can be considered as a crucial factor in the choice of public key systems to use digital signatures, especially for smaller devices with less computational resources [70]. Nonetheless, in some cases it is not necessary to generate RSA keys for each use. For this situation, we would remark that the problem mentioned before is not so dramatic, since RSA is comparable to ECC for digital signature creation in terms of time, and it is faster than ECC for digital signature verification [71].

Furthermore, we could incorporate Merkle trees [72], such that we will reduce the number of digital signatures to handle. Actually, as a real example which uses Merkle trees we have the well-known system Bitcoin[4].

3.2.4 Challenge 4: Availability

Users of cloud environments typically want to place a large amount of data in cloud servers. Some of this information might not be used during long periods of time, but it must be available when required. CPs check data availability by storing files with redundancy or error correction. Although the main cloud storage providers deploy hardware redundancy mechanisms, from a general point of view, users cannot be sure whether their files are vulnerable or not against hard drive crashes [73].

3.2.4.1 Solutions

Despite high adoption rate among consumers, cloud storage services still suffer from many functional limitations and security issues. Recent studies propose the utilisation of RAID-like techniques in addition to multiple cloud storage services as an effective solution. Therefore, as a simple solution, the user could think about splitting her data between different CPs, that is, saving a copy of each her files in each CP. In this context, if one of these CPs would have a critical problem in its servers, the user could access all her files stored in other providers. However, the management of files stored in multiple cloud storage servers is not an easy task.

Fortunately, recently some cloud storage managers has been proposed to handle cloud storage in multiple SPs [74]. Moreover, in [75] it is proposed a solution for mobile devices that unifies storage from multiple CPs into a centralised storage pool, which represents a clear improvement in terms of availability, capacity, performance, reliability and security.

3.2.4.2 Limitations

The solutions presented in [74] are private, so the user would have to pay a fee each month if she would want to use this service without limitations. Therefore, it is for this reason that these solutions have no place in our FCS solutions mindset. Additionally, using these proposals the user could not feel comfortable, since she would let another third-party company manage her files. In this case, the user would have to read each company's term of services to see how they manage her files and she would have to trust them.

[4] https://bitcoin.org/en/

3.2.5 Challenge 5: Data location

The location of user data in the cloud can be a critical security issue. When the provider guarantees that the data are stored within specific geographic area, users might not have the assurance about this fact. On this point, it is very important to consider the recently decision of the European Court of Justice made on the last October of 2015: the annulment of the EU-US data sharing agreement named Safe Harbour [76]. This revocation prevents the automatic transfer of data of European citizens to the United States of America (USA). This is a critical decision, since most SPs are from the USA.

3.2.5.1 Solutions

Cloud service latency is a hint to infer the geographic location of data as stored by CPs. Indeed, it will take more time in their transmission as they are farther from user location. These measures have to be carried out with high degree of precision since the information moves really quickly in electronic communications.

An example of location proof is the use of distance bounding protocols [77]. These protocols always imply a timing phase in which the verificator sends a "challenge" and the provider responds. The provider is required to respond within a time limit which depends on the distance between the provider and the user. Cryptographic authentication procedures are used to ensure that the responses come from the intended provider instead of other entity that says to be the provider.

In the cloud, this method can be used to verify with accuracy the continent, and even the country where data are stored.

3.2.5.2 Limitations

In the proposed method there are several uncertainties that can complicate this measure, like a delay in the Internet communication. Furthermore, there can be some delays caused by the time that the server spends in order to access the disk and to the correct sector in which the blocks of data are stored. In fact, this measure is really dependent of the storage service, since for example Dropbox has to decrypt the information before sending it (as the encryption/decryption is made in the server), whereas this is not necessary in Mega, since this operation is made in the client.

3.2.6 Challenge 6: Data deduplication

With the huge potential storage space offered by CPs, users tend to use as much space as they can, so providers constantly look for techniques which can minimise redundant

data and maximise free space available. One adopted technique is deduplication. The simple idea behind deduplication is to store only once deduplicated data (both files and blocks). Therefore, if a user wants to upload a file (block) that is already stored, the CP will add the user in the list of owners of this file (block). Deduplication is able to save space and costs, so that many CPs are adopting it.

On the other hand, some users require data protection and guarantees of confidentiality through encryption. Unfortunately, deduplication and encryption are two technologies that come into conflict between them. While the objective of deduplication is to detect data segments that are identical and to store them only once, the result of encryption is to make indistinguishable two identical data segments. This means that if data are encrypted by the user in a standard way, the cloud storage provider could not apply deduplication since two identical data segments will be different after their encryption. On the other hand, if data are not encrypted by users, the privacy cannot be guaranteed and data will not be protected against *curious* CPs [78].

3.2.6.1 Solutions

One solution that has been proposed in order to solve these two conflicting requirements is the use of Convergent Encryption (CE) [79]. This technique consists of using as encryption key the hash of the content of data we want to encrypt, such that two equal files will generate the same encrypted file. This will allow both deduplication and assets encryption [80].

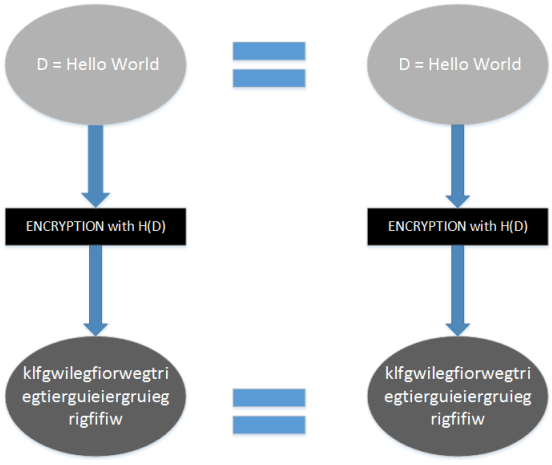


Figure 3.2: Example of CE.

Apart from this solution, there are other alternatives which may solve the problems aforementioned, like the ClouDedup tool [78]. This solution proposes the inclusion of

an intermediate server between the users and the SPs in the cloud. Users will send to the intermediate server their blocks encrypted with CE, along with their keys, so that the server is able to identify duplicate blocks between all blocks which are sent by different users. Afterwards, the server will be in charge of deduplication and management of a second non-convergent encryption which confirms the security of the process, sending to the storage service only blocks that are not duplicated.

3.2.6.2 Limitations

First of all, CE suffers of several weaknesses, including dictionary attacks. Furthermore, it adds a privacy risk: with CE in the client side, a malicious user could know if it already exists some particular information in the cloud, which could be not acceptable for some users.

However, one possible solution that could counteract the weaknesses of CE is taking in account the popularity of data segments. The data segments which are stored by several users, that is, the most popular, are protected by the weak mechanism of CE while unpopular data segments, which are unique in the storage, are protected through symmetric encryption scheme using a random key, which provides the highest level of protection while improves the computational cost in the client. This approach allows an efficiently deduplication since popular data segments that are encrypted with CE are the one that need to be deduplicated. This scheme also assures the security of data, since sensitive data, less popular, have a great protection with semantically-secure encryption while popular data segments do not suffer the weaknesses of CE since they are less sensible as they are shared by several users [81].

Nevertheless, this approach has another challenge: users have to decide about the popularity of each data segment before its storage. The focus of the schemes based on popularity is on the design of secure mechanisms in order to detect the popularity of data segments [81]. Thus, in this schema data need different levels of protection depending of their popularity, so a data segment could be popular when it belongs to more than t users (where t is the popularity threshold). The popularity of a block is seen as a trigger for its deduplication. In the same way, a data segment would not be considered popular if it belongs to less than t users. This is the case of highly sensitive data, which are unique and they do not have to be deduplicated. When data segments that are unpopular begin to be popular, this is, threshold t is reached, the encrypted data segment is converted in its form of convergent encrypted in order to enable its deduplication. Therefore, when a user wants to upload a data segment, she must first discover the popularity degree of data in order to perform the appropriate encryption operation.

Finally, as far as we know, the best solution proposed is the tool called ClouDedup [78], which in its architecture, apart from the basic storage server, has a metadata manager and an additional server. This last server adds an additional non-convergent encryption layer on the most sensitive data in order to prevent well-known attacks against CE and to protect data confidentiality. On the other hand, the metadata manager is responsible of the key management task, since block-level deduplication requires to memorise a large number of keys. Therefore, deduplication is performed at block-level and an efficient key management mechanism is defined in order to avoid that users have to store a key for each block.

3.2.7 Challenge 7: Version control of encrypted data

If the upload of encrypted files to the cloud is allowed, there exists the problem of the version control of encrypted data. Version control of plaintext files is trivial, since it is possible to check which part of the original file has been modified. In information encryption, it is a must to use a secure cipher mode as for example CBC [82], in which an isolated modification in the original file modifies the whole encrypted text produced (condition that is essential in all cipher methods). However, these cipher modes make that the version control system cannot be able to know what specific part has been modified, having to store each file several times, one for each file version. This increases the space that is needed for the repository and reduces the efficiency.

3.2.7.1 Solutions

An approach which has proposed recently is to divide the files uploaded to the cloud in data objects [83]. Each data object has a variable size with a maximum configurable size. If one file has a smaller size than this maximum, then it can be represented by only one object; otherwise, it has to be divided into multiple objects. As a result, if some modifications are produced in a big file, the user only would upload the objects that have been modified, instead of the whole file, saving costs on both assets upload and storage.

In many cases, the same object can appear in multiple backup versions, or different objects can have the same content in the same or different versions. Therefore, if two objects have the same content, then it is only necessary to store one object in the cloud and to create small pointers in order to reference these objects. For determining if two objects are identical, hash functions can be applied.

In this fashion, the user could place different backups at different times, and organise these backups in different versions. For each version, there would be a

metadata object that describes data objects. Figure 3.3 shows how different backup versions are uploaded. Suppose that at time t_1, we want to upload the version V_1 that has four objects: (O_1, O_2, O_3, and O_4). Imagine that after, in the instant $t_2 > t_1$, O_1 and O_2 are not included, and O_5 and O_6 are added. Therefore, the new version V_2 is going to upload physical copies of O_5 and O_6, and its metadata objects have pointers that reference to the physical copies of O_3 and O_4 in the version V_1. Finally, all metadata objects and data objects are stored in the cloud. This being the case, it would have a different encryption key for each data block. Consequently, if a modification on the version V_1 is done, it is required to upload only the block that has been modified.

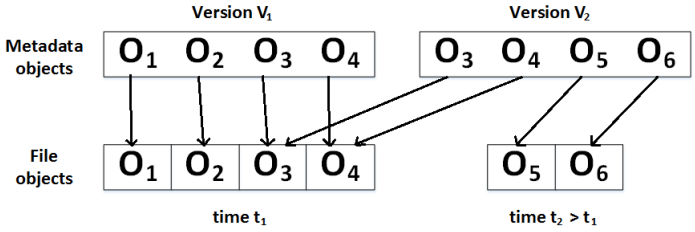


Figure 3.3: Control version mechanism.

3.2.7.2 Limitations

The solution which consists of dividing the file in different chunks would be a good option. However, it has the problem of higher management costs for the user, since she has to be in charge of splitting the files she uploads and to compare each block in order to perform an appropriate version control. Moreover, in case of applying this solution, the main problem would be the management of a large number of encryption keys, one for each data object stored in the cloud [83].

3.2.8 Challenge 8: Assured deletion of data

The assured deletion of data guarantees that files are inaccessible once user has requested to CPs to delete them. It is not desirable to keep data backups permanently, since sensitive information can be exposed in the future due to a security breach or a poor management made by CPs. In order to avoid these problems, most companies hold their backups for a finite number of years. After that, they request an assured deletion of their data [83].

This deletion needs to be provided for the users to actually destroy their data backups under request. Moreover, CPs can use multiple copies of data over cloud infrastructure to have a great tolerance against failures. As providers do not publish

their replication policies, users do not know how many copies of their data are in the cloud, or where those copies are stored. Hence, it is not clear how providers can delete all the copies when users ask for removing their data. In the particular case of Dropbox, when a user removes a document on this platform, this file is queued for permanent deletion. Queuing is used to enable file recovery and version history [84], but it could enclose a security risk.

3.2.8.1 Solutions

There are different ways to get assured deletion of data, but the best approach is to use cryptographic protection, since if the user destroys the keys which are necessary to decrypt her data blocks, then these blocks will be inaccessible [83].

3.2.8.2 Limitations

The cryptographic protection can have a main limitation, since we must assume that decryption keys are maintained by a key management system that is totally independent of the cloud system and it is totally controlled by users, which is a great overload for the latter. Therefore, the same limitations explained in Section 3.2.2.2 can also apply for this section. On the other hand, note that client-side encryption only protects users' data against CPs, but not against changes in the users groups in which is performed a key distribution protocol. In this specific case, if a user is revoked from a group, she could still access all previous versions of shared files that belongs to this group if she has previously stored them locally.

3.2.9 Challenge 9: API's validation

The current technological scenario has been called by many as the era of containers (*container age*), since a large number of initiatives are based on cloud solutions built by third-party software products. Thus, it has emerged a trust model that is not always taken into account when analysing the security risks of our proposals. Indeed, many of these third-party solutions are given by APIs that have not always been properly validated in terms of security.

In order to illustrate this type of security concern, we would want to highlight a significant security problem which was found by a research group from the Worcester Polytechnic Institute [85], showing that they could obtain secret cryptographic keys that were in a virtual machine of Amazon Web Service (AWS), used for encrypting sensitive data. This attack was based on execute in virtual machines of AWS cloud a vulnerable library that is called Libgcrypt, from which they were able to extract a

great quantity of information, that contained the secret encryption key. We have to mention that this bug was fixed in February 2015. As solutions for this problem, we have to remark that there are several steps that users can take in order to protect themselves against vulnerabilities, like using updated software and installing all Open Source software patches as soon as they are available.

In October of 2015, another vulnerability was found by a Chinese developer [86], who said that Outlook.com, OneDrive and Microsoft account pages incorporated a unique user identifier in the URLs that is known as CID. The CID was a 64-bit integer which was associated with each Microsoft account, and it was used in Microsoft APIs for user authentication. As a consequence of the underlined security problem, anyone who was able to monitor DNS traffic, or had access to user web traffic, could see this id, since the CID was included in the host name part of the URL.

Due to the high complexity of these systems mentioned above, first, is highly recommended the use of a Secure Development Lifecycle (SDL). This methodology provides best practices for developing secure applications, with the aim of overcoming the challenges of making software secure [87]. SDLs are the key step in the evolution of software security and have helped to bring attention to the need to build security into the Software Development Life Cycle (SDLC).

Along with SDLs, since there are several details to take into account in a software development, developers should rely on automatic tools. Indeed, the application of formal methods to verify security properties has received more repercussion lately in order to avoid this type of security holes [88]. Several methodologies support the application of these mechanisms and theories with the aim of get high security guarantees for critical systems and protocols. However, a straight application of these tools can involve a high consumption of resources during the early stages of design, since some faults can be detected by simple performing an informal analysis [89].

In this aspect, an intelligent approach that a developer could take is to add an extension over some public cloud storage services which already exist, in order to counteract the weaknesses that these services have. Then, she will only have to validate her extension, since these public cloud storage services have already been validated by the security community.

3.2.9.1 Solutions

A methodology for the design of secure protocols is proposed in [89], which covers the whole design process and takes into account the problems mentioned before, following an iterative approach that consumes fewer resources in the early stages than in the following stages. As a consequence, the complexity of the failures detected increases as

we progress through the methodology steps. Furthermore, a correct abstraction of the protocol's environment and attacker model are made. In this vein, the methodology proposed allows a fine control in the attacker modelling in order to fit it to the specific context in which the protocol is going to be executed.

Therefore, we have to remark the necessity of an adaptive process with feedback when any protocol of information security is designed. Any possible methodology must be built over evaluation tools for the objectives and acceptances which are involved in design and implementation stages, and any threat identified during the implementation or production should be managed as feedback for future improvement in the design of the system [90].

On the other hand, we have to highlight companies as Cryptosense[5], which is a recent start-up which is looking to commercialise techniques for analysing the security of APIs.

3.2.9.2 Limitations

Nevertheless, the human factor also takes part in the analysis of protocols procedure, so a bad formalisation can make that some errors are not detected. Then, as in any engineering process, the solution presented does not provide the 100% of success, but it helps avoid a great quantity of failures.

3.2.10 Challenge 10: Usable security solutions

Daily activity of today citizens is very dependent on cloud services. Nevertheless, this use is not always a secure as it should be. The incorporation of security measures is consequently required, but the related changes should not erode users' acceptance of the so-modified cloud services [91, 92]. In other words, secure cloud services solutions should be also easy to use.

3.2.10.1 Solutions

The so-called security-by-design and privacy-by-design paradigms [93] are hot topics in cryptographic engineering. On this point, we have to consider standards [94] and well-known technologies as a basic framework for designing new security systems.

Furthermore, after the security solution is developed and before it is published for its use, its usability has to be tested. For example, in [91] it is proposed to evaluate a specific security solution, PGP 5.0, through a cognitive walkthroughs, a heuristic evaluation and a laboratory user test, whose combination produces an exhaustive evalua-

[5] https://cryptosense.com/

tion. In [95] are provided Human-Computer Interaction (HCI) methods and practices for each phase of the SDLC, in order to create usable secure and privacy-respectful systems. Moreover, there are recent efforts on formalising and automatically assessing the role of human factors in security protocols [96, 97].

3.2.10.2 Limitations

Firstly, a key limitation to consider is that security and privacy are rarely the user's main goal, and they would like privacy and security systems and controls to be as transparent as possible [95]. On the other hand, users want to be in control of the situation and understand what is happening. As a consequence, these two factors make harder to develop usable security applications.

Finally, in order to be successful in the development of usable security solutions, it is needed to have a multidisciplinary team including experts from the fields of IT, information science, usability engineering, cognitive sciences and human factors. This could suppose a main limitation, since no all companies have enough resources in order to form this kind of heterogeneous work group.

Chapter 4

Encrypted Cloud

4.1 Introduction

As it has been mentioned in Chapter 1, the main objective of this project is to implement an Open Source desktop single-page application, called Encrypted Cloud. This application is functional in both Ubuntu and Windows, and allows to upload, download and share any type of encrypted file placed on FCS. Encrypted Cloud has been designed taking into account the most relevant challenges that we have previously underlined in Chapter 3.

In this project, the main feature is that the user can choose a variable number of local directories where she wants to place her encrypted files. Hence, the application acts as a middleware showing the selected directories as a single unified directory. This being the case, the user can encrypt and distribute the files in a transparent way. These files are added to her profile through the application. In fact, a user who have installed on her computer different synchronisation folders of different CPs could select them in Encrypted Cloud. Henceforth, the user can encrypt her files and store them in any of these CPs, taking advantage of the free cloud storage space that each CP provides. She can also share encrypted files that are stored in local directories that were previously shared. To do this, we implemented our own cryptographic key distribution protocol. In addition to this core functionality, there were defined a set of services that allow to detect deletion and movement of files by unauthorised third parties. Furthermore, the design of the application has been made under the premise of achieving a definition as flexible as possible, allowing to include more local directories into Encrypted Cloud without being altered any other functionality. This flexibility has the great advantage that the application can use other types of local directories such as network drives.

As a summary, the first main objective pursued with this solution is to leverage

the advantages that FCS solutions provide, unify the free space offered by different providers, and add a security layer. Encrypted Cloud has been conceived as a way to enlarge SMEs' confidence in placing their information assets stored onto third-party cloud platforms. In this sense, it is necessary to underline the cross-platform nature of the proposed solution and the possibility of adding new features under the Open Source application created.

The next section presents the planning followed to achieve such a goal. After that, it is provided the list of functional and non-functional requirements that Encrypted Cloud must fulfill. The concretion of the application's requirements is properly complemented with the description of the relevant aspects about the design, functionality and implementation of Encrypted Cloud. Since in any engineering project it is not possible to assert the achievement of the original objectives without a thorough evaluation, the last part of this chapter is dedicated to explain the test plan carried out to validate Encrypted Cloud.

4.2 Planning

This section details through a Gantt chart the planning followed throughout the project (see Appendix A). The diagram follows a labour calendar, dedicating an average of three hours per day, and it is composed of five stages:

Documentation (02/10/15 - 21/12/15)

During this first phase of the research work it was accomplished a review of the main papers and projects related to information security in cloud environments, which is a relevant topic that is being currently addressed in both academia and the business world. It was also required a review of similar technologies for the sake of the project's concretion. Besides, the line of the work proposed demanded to study the issues of privacy and confidentiality in cloud environments, as well as the integrity control for information assets that the user stores in her CP. The usability of the final product has been another key objective in this project.

As part of the evaluation of cloud security solutions in the corporate domain, in October of 2015 I attended to the EMC Forum 2015[1], a congress where large companies which work with cloud computing show their latest solutions in this technology and their main concerns. In most of the cases, they claimed that the main concern was the issue of the insecurity of its assets in the cloud. From these speeches, I could extract different ideas which were really helpful for the analysis of the requirements

[1] www.emc.com/es-es/events/forum/digital-es.htm

to be satisfied by Encrypted Cloud. Finally, it should be noted that this work of investigation has resulted in a research paper called "*Combining usability and privacy protection in public cloud storage servers: review of the main threats and challenges*" (still under preparation), and which is the bottom line of Chapter 3.

Analysis and Design (22/12/15 - 07/01/16)

During this phase it has been carried out the specification of the requirements of the application, as well as a detailed study of its design. It should be emphasised that in both design and implementation phases, the security and privacy has been treated as fundamental requirements.

Programming (08/01/16 - 01/04/16)

Once the design phase was finished, the next step was to start with the application development. In this stage, an important aspect was the decision about the technologies to build up the application. In order to have a large number of users, the best option was to implement a cross-platform desktop application. This aspect along with the high demand of full stack developers in the current job market, resulted in the decision to use a new technology called NW.js (formerly known as node-webkit). Therefore, a learning phase of this technology was needed, and more specifically with Node.js. Once I finished two MOOCs[2][3] on this programming language, the following step was to study the cryptographic API of Node.js[4], in order to create some encryption examples with it. After that, the encryption of non-shared files was the first functionality developed. Then, the option of sharing documents with other users was added, for the particular case in which these files are stored in shared folders. Finally, the UI and other functionalities like a service which monitors unauthorised deletion or movement of files were implemented.

Tests (04/04/16 - 29/04/16)

The main objective of the testing phase was to check if Encrypted Cloud satisfied the requirements established. For this, a test plan was designed (see Section 4.7), performing both unit and integration tests and documenting each result.

Final report (02/05/16 - 31/05/16)

The last stage was the writing of this document, which includes all the steps that were taken to implement the application, as well as all the necessary details to understand how the application works.

[2] www.miriadax.net/web/javascript-node-js
[3] www.edx.org/course/introduction-mongodb-using-mean-stack-mongodbx-m101x
[4] www.nodejs.org/api/crypto.html

4.3 Requirements Catalogue

This section presents both the functional and non-functional requirements that Encrypted Cloud must fulfill.

4.3.1 Functional Requirements

Functional Requirements (FRs) document the operations and activities that a system must be able to perform.

FR1. Encrypted Cloud set up.

In the first start of Encrypted Cloud, the user must provide her password twice, the asset distribution protocol and at least one non-shared local directory.

FR2. Encrypted Cloud authentication process.

Encrypted Cloud does not have a registration process for new users, because it is a mono-user application. However, a password is necessary for the encryption/decryption of non-shared files and for keeping encrypted the local database. Therefore, the user has to provide it each time that she starts again the application.

FR3. Arbitrary number of local directories.

The user can choose several directories, which can be shared or not. The user stores her assets in those folders in a transparent way. At least, she must select one non-shared local directory.

FR4. Move a local file previously encrypted to a local directory selected by the user.

The user can move files from a local folder to a local directory previously selected. These files can be dragged into the desired destination or they can be selected through a local file explorer. Multiple files upload is also supported. In any possible case, the application encrypts all files before storing them in these local directories, with the aim of protecting their confidentiality and privacy.

FR5. Delete a file previously stored in an application's local directory.

The user can select and delete files that were previously added and stored through the application.

FR6. Decrypt and view a file previously added by the user to the application.

If the user makes a double-click on a file handled by Encrypted Cloud, this file will be decrypted and displayed using the suitable application. The file selected is saved in a local temporary folder (called *tmpFiles*) where it is decrypted and opened for its visualisation. When the application is closed, all files which are in this temporary folder are deleted. Therefore, all files stored in the local paths selected in the set up of Encrypted Cloud are always encrypted as long as they are in these folders.

FR7. File integrity verification.

Each file is digitally signed by the user who uploads it to the application. When this or other users access this file, they can check this digital signature in order to know whether this file has been modified or not by unauthorised parties.

FR8. Monitoring of files deleted or moved by unauthorised parties.

There is one asynchronous task which is in charge of monitoring all files previously uploaded by the user, in order to know it they are deleted or moved by unauthorised parties.

FR9. Create a folder.

The user can press the "Create Folder" link in the left panel of the application and then she will have to provide the folder name, in order to create a new folder in the current directory of the application.

FR10. Delete a folder.

The user can delete a folder by selecting it in the application and then pressing the "Delete Folder" link in the left panel of the application.

FR11. Asset distribution protocol.

In both folder creation and file upload through non-shared directories, Encrypted Cloud provides four asset distribution protocols to determine where these elements will be stored:

1. **Rotating:** a new asset is saved in a different non-shared directory each time.
2. **Random:** a new asset is stored in a folder randomly selected among those non-shared directories that were chosen by the user.

3. **Space Used:** a new asset is saved in the non-shared directory with more size (MB) available.
4. **Availability:** a new asset is stored in all non-shared directories with the aim of having the highest asset availability.

FR12. Transparent unification of the storage provided by the local directories selected.

The application unifies all paths selected by the user as if they were only one directory. The application saves each file in one or several of these local directories, and this file distribution policy depends on the preferences chosen by the user in the set up of the application. Moreover, if the user selects as local directories the local folders of cloud desktop applications as Dropbox, Google Drive, Mega, etc., she could unify the free cloud storage provided by them in Encrypted Cloud in a transparent way.

FR13. Join a shared directory.

The user can join a shared directory which was previously selected by her in the set up of the application, choosing first the shared directory and then pressing the "Join Folder" link in the left panel.

FR14. Leave a shared directory.

If a user was previously joined to a shared directory, she could leave it whenever she wants, choosing first the shared directory and then pressing the "Leave Folder" link in the left panel of the application. For both the join and leave folder actions, it is used our own protocol which is in charge of managing all the operations with shared directories.

FR15. Share an encrypted file.

The application allows a user to share encrypted files with other users, using for this our own protocol for symmetric key distribution (see Section 4.5.4.2). This sharing is performed in these local directories which are previously selected by the user in Encrypted Cloud and which are already shared with other users.

FR16. The database of the application is encrypted.

The local database of Encrypted Cloud is encrypted. The encryption key is derived from the password required to the user in the authentication process.

FR17. Local backup.

Encrypted Cloud has an option for making a local backup of all files uploaded. For this, the user has to press the "Local Backup" link in the left panel. Then, she has to select a local directory in which she wants to save this backup, but this directory must not be the same that those selected in the set up of Encrypted Cloud.

FR18. Reset application.

The user can reset the application whenever she wants, pressing the "Reset App" link in the left panel of the application. At this moment, all application settings and files previously uploaded will be deleted. However, before performing this action, the user has the option to make a local backup of all files stored in Encrypted Cloud.

4.3.2 Non-Functional Requirements

Non-Functional Requirements (NFR) define features of the application which are not necessary for its working.

NFR1. Export local database.

The application allows the user to export the local database into a local directory. This functionality is necessary if the user has reset the application and in the future needs to recover her keys from a previous database or in the case in which the current database is corrupted.

NFR2. Import local database.

The application allows the user to import a local database from a local directory.

NFR3. Password modification.

The user can change her password through a link called "Preferences" located in the left panel. After that, she has to provide both current and new passwords.

NFR4. Cross-platform application working on Ubuntu and Windows.

NFR5. Modularity of the source code of the application.

The source code of the application is separated in different libraries, classified by their functionality provided in the application. This approach can help to reuse some modules in other applications.

NFR6. Usability.

Encrypted Cloud's functionalities are really similar to other file explorers, so all these users that use this kind of applications will not have problems with Encrypted Cloud.

NFR7. Code documentation.

The application is documented in detail, which eases its maintenance and its integration with other systems.

4.4 Design

This section presents two essential diagrams in the design phase of Encrypted Cloud.

4.4.1 Entity-relationship database diagram

An entity-relationship database diagram is a graphical representation of an information system that shows the relationship between different entities. The entity-relationship diagram designed for Encrypted Cloud is represented in Figure 4.1.

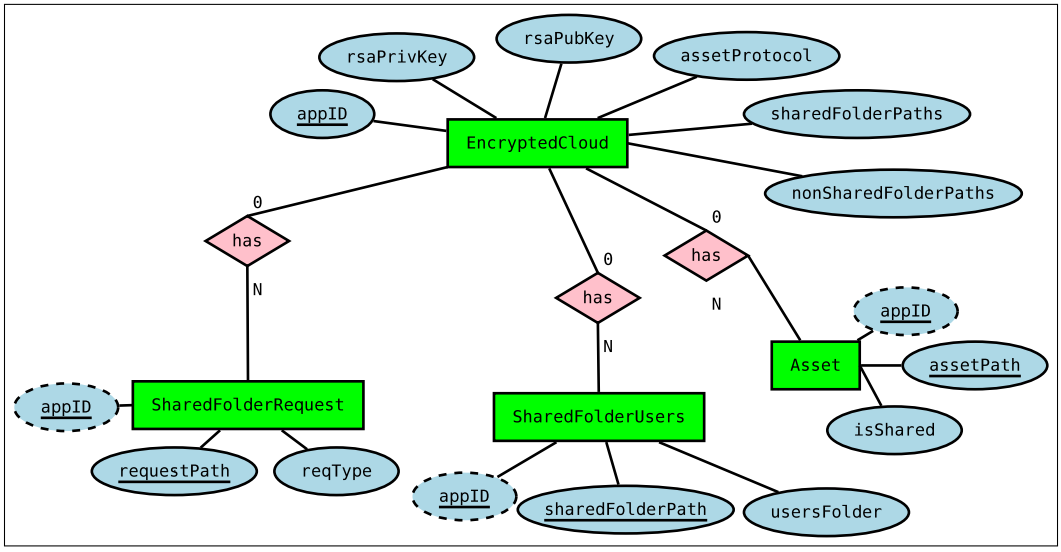


Figure 4.1: Entity-relationship database diagram.

This database has four different entities. The following figures show one register per each entity obtained from a decrypted database.

The first entity is **EncryptedCloud**, that stores six attributes selected in the set up of Encrypted Cloud. The first one is *appID*, a random integer that identifies the application. This id is the primary key, and it is used as part of the file name of the

join/leave shared folder requests (see Section 4.6.2.3.1 and Section 4.6.2.3.2) and in the folder name where the local backup of the application is stored (see Section 4.6.2.3.13). There are also two strings, *rsaPrivKey* and *rsaPubKey*, which represent the RSA key pair that is only generated in the set up of Encrypted Cloud. This key pair is necessary for performing digital signatures (see Section 4.5.4.1.3) and sharing a symmetric key (see Section 4.5.4.2). Therefore, after doing this set up, in the next time in which the user starts Encrypted Cloud, it is only needed to get these keys from the database. In the first start of Encrypted Cloud, the user has also to provide both shared and non-shared local directories where she wants to store her assets. These two attributes are stored as strings separating each path with a ";" as a delimiter. Finally, *assetProtocol* is the attribute that decides in which non-shared folder will be placed each asset. Each protocol is represented by its name except the "Rotating" protocol, where it is inserted the index of the non-shared folder in which was stored the last file or folder. The Figure 4.2 shows one register of the EncrpytedCloud entity.

appID	rsaPrivKey	rsaPubKey	assetProtocol
61729	-----BEGIN RSA PRIVATE KEY-----...	-----BEGIN RSA PUBLIC KEY-----...	0

sharedFolderPaths	nonSharedFolderPaths
/home/alejandro/Dropbox/SharedFolder;/home/alejandro/Desktop/Fake Shared	/home/alejandro/Dropbox;/home/alejandro/MEGA

Figure 4.2: Register in EncrpytedCloud entity.

The second entity is **Asset**, which stores each asset path (both folders and files) uploaded to Encrypted Cloud. The main goal of this entity is to check if these assets have been deleted or moved by unauthorised parties. The primary key is composed by *appID* (foreign key of EncryptedCloud entity) and *assetPath*, the path of each asset uploaded. The other attribute is *isShared*, a boolean that indicates if the asset is shared or not. The Figure 4.3 shows one register stored in Asset entity.

appID	assetPath	isShared
61729	/home/alejandro/Dropbox/tatras.jpg	false

Figure 4.3: Register in Asset entity.

SharedFolderUsers entity stores the users joined to each shared folder of Encrypted Cloud. After the join and leave requests has been properly executed, the registers of this entity are updated each time that each shared folder is updated by its owner (see Section 4.6.2.3.3). This entity is needed for the case in which it is improperly removed or modified the file that contains the list of the users joined to each

shared folder. The primary key of this entity has two attributes: *appID* (foreign key of EncryptedCloud entity) and *sharedFolderPath*. The third attribute, *usersFolders*, stores all RSA public keys (separated by ";") of each user joined to the shared folder. The Figure 4.4 shows one register stored in SharedFolderUsers entity.

appID	sharedFolderPath	**usersFolder**
61729	/home/alejandro/Dropbox/SharedFolder	-----BEGIN RSA PUBLIC KEY-----...

Figure 4.4: Register in SharedFolderUsers entity.

The last entity is **SharedFolderRequest**, that is in charge of storing all join or leave requests that the user has made with regard to the shared folders of Encrypted Cloud. Each time that the user makes a join or leave request to a shared folder, it has to be managed by the owner of this shared folder. Therefore, this entity saves each request, with the aim of checking if these requests have been completed. As a result, this entity also confirms that a request has not been deleted by a malicious server. The primary key of this entity is composed by two attributes: *appID* (foreign key of EncryptedCloud entity) and *requestPath*, the full path where it is stored each request. The third attribute is *reqType*, which allows to distinguish between join and leave requests. The Figure 4.5 shows one register of the SharedFolderRequest entity.

appID	requestPath	**reqType**
61729	/home/alejandro/Desktop/Fake Shared/.join_requests/.61729_join.txt	join

Figure 4.5: Register in SharedFolderRequest entity.

Related to the relations between each entity, we should recall that the application (**EncryptedCloud**) can have from zero to N **Assets** uploaded, it handles from zero to N requests over a shared folder (**SharedFolderRequest**), and it also has from zero to N lists of users associated to shared folders (**SharedFolderUsers**).

4.4.2 Graphic diagram

The graphic diagram of the application allows to display in a graphical and easily way the navigability between the screens of the application. Remind that Encrypted Cloud is a single-page application with the main goal of providing a fluid user experience. Therefore, it is possible to explain with only three main screens its structure as shown in Figure 4.6.

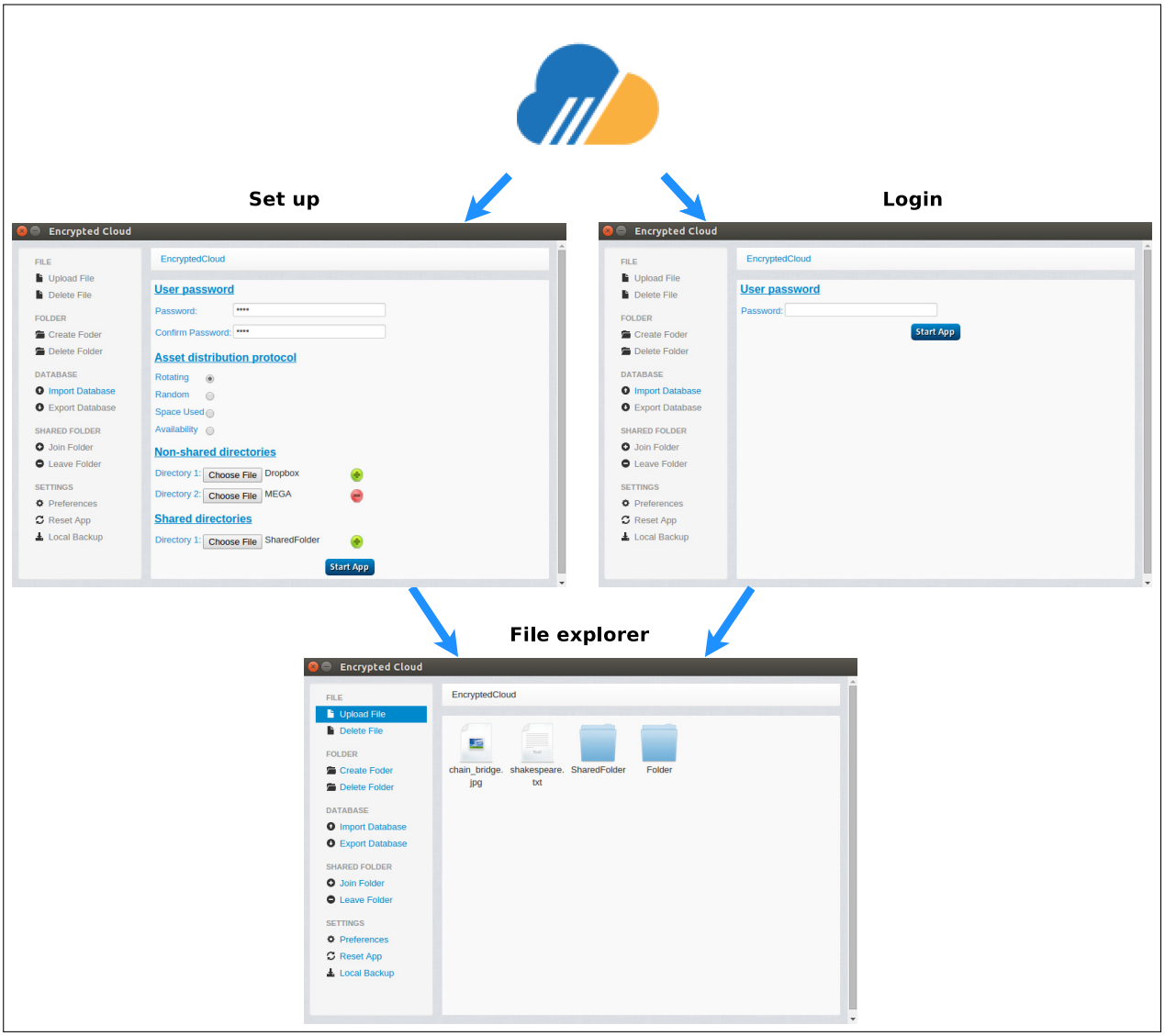


Figure 4.6: Graphic diagram.

When it is the first time the user starts Encrypted Cloud, she has to provide different settings: her password, the asset distribution protocol, with which will be decided where the files and folders will be stored and both shared and non-shared directories. Figure 4.7 shows the form that the user must fill out to establish the related settings.

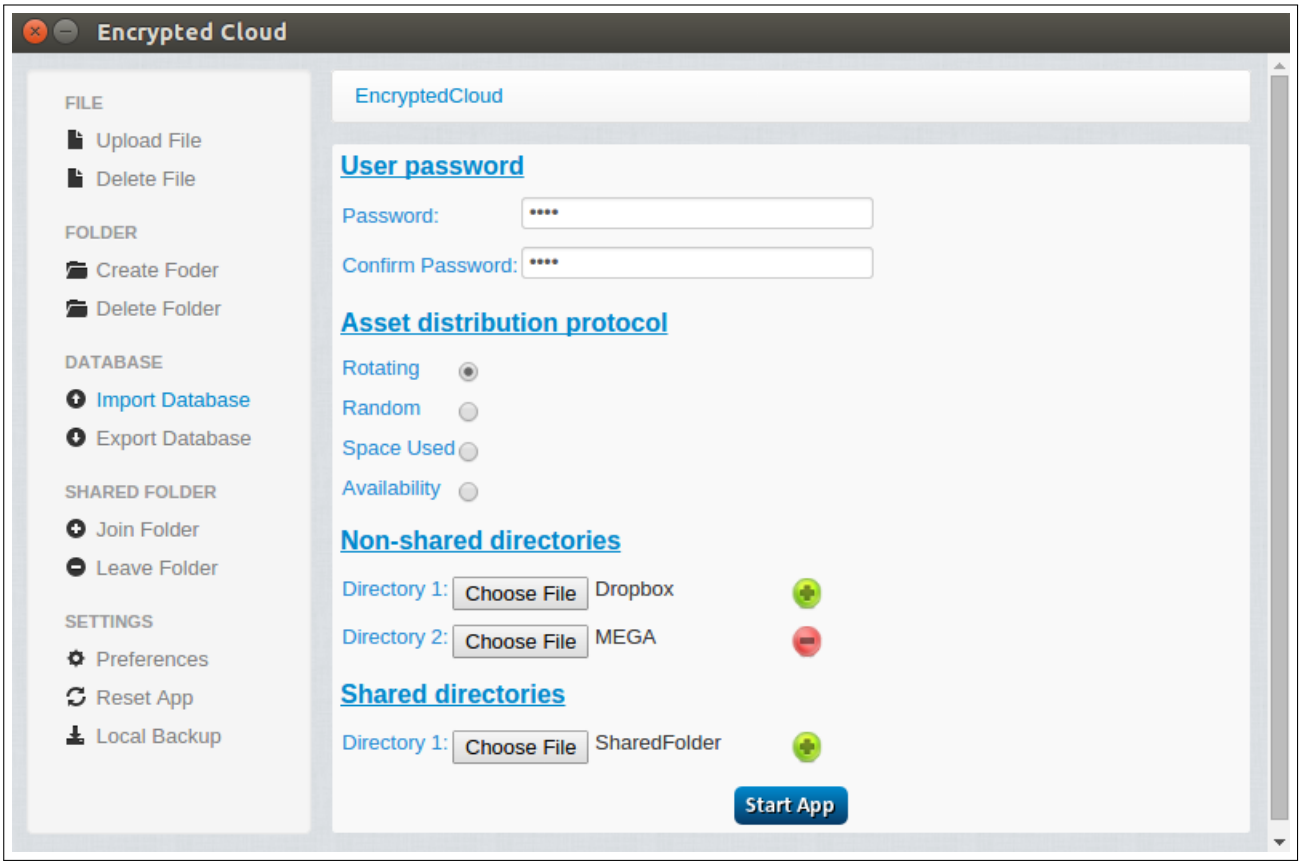


Figure 4.7: Encrypted Cloud set up.

Otherwise, if it is not the first time the user starts Encrypted Cloud, she only will have to provide her password (see Figure 4.8).

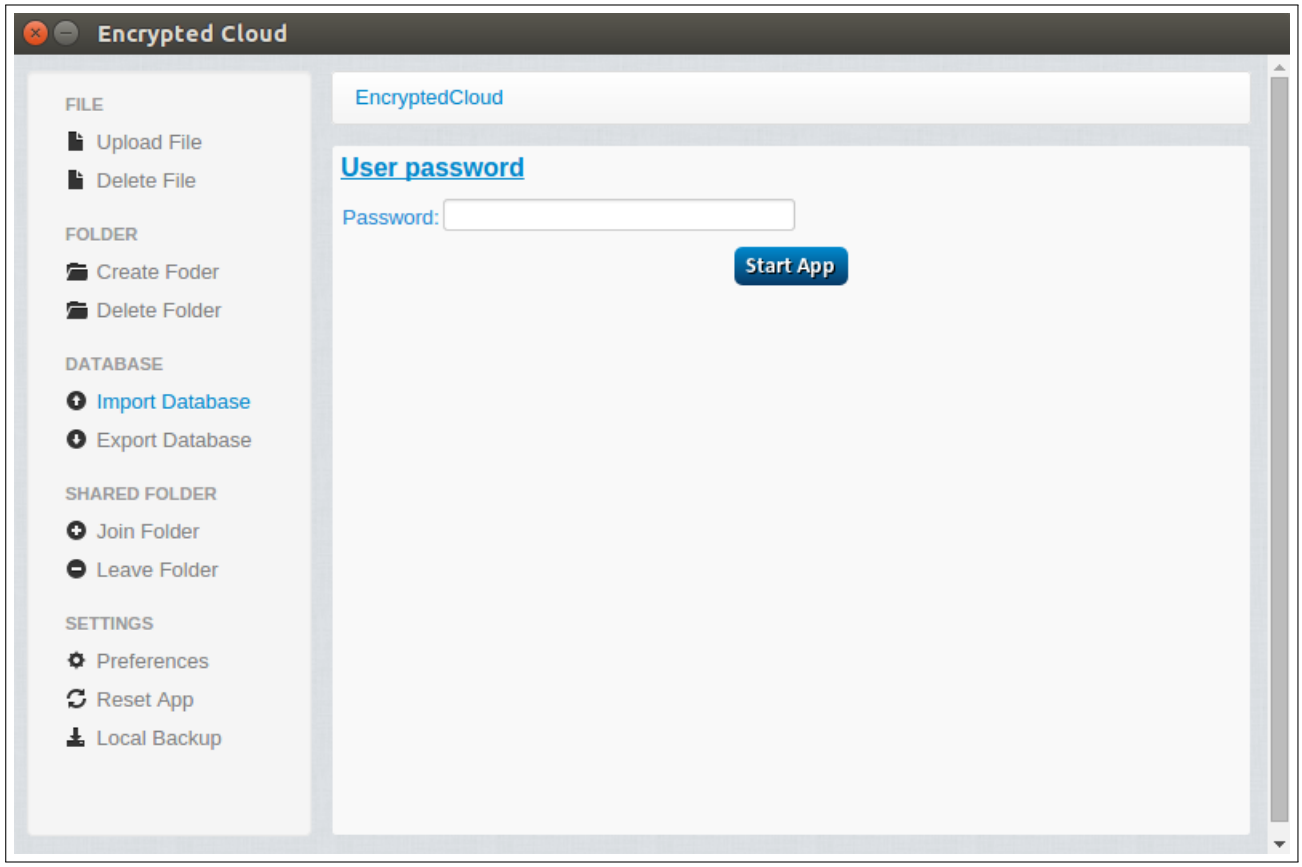


Figure 4.8: Encrypted Cloud login.

Finally, in both cases after the Encrypted Cloud set up or login, the screen that appears in Figure 4.9 will be loaded. This window is divided into two main parts, and it is based on a file explorer from a GitHub project[5]. First, there are eleven options in the left panel that are explained in detail in Section 4.6.2. On the other hand, the central part of this screen is the file explorer, which displays all the elements that are contained in both shared and non-shared folders (see Figure 4.9).

[5] https://github.com/zcbenz/nw-sample-apps/tree/master/file-explorer

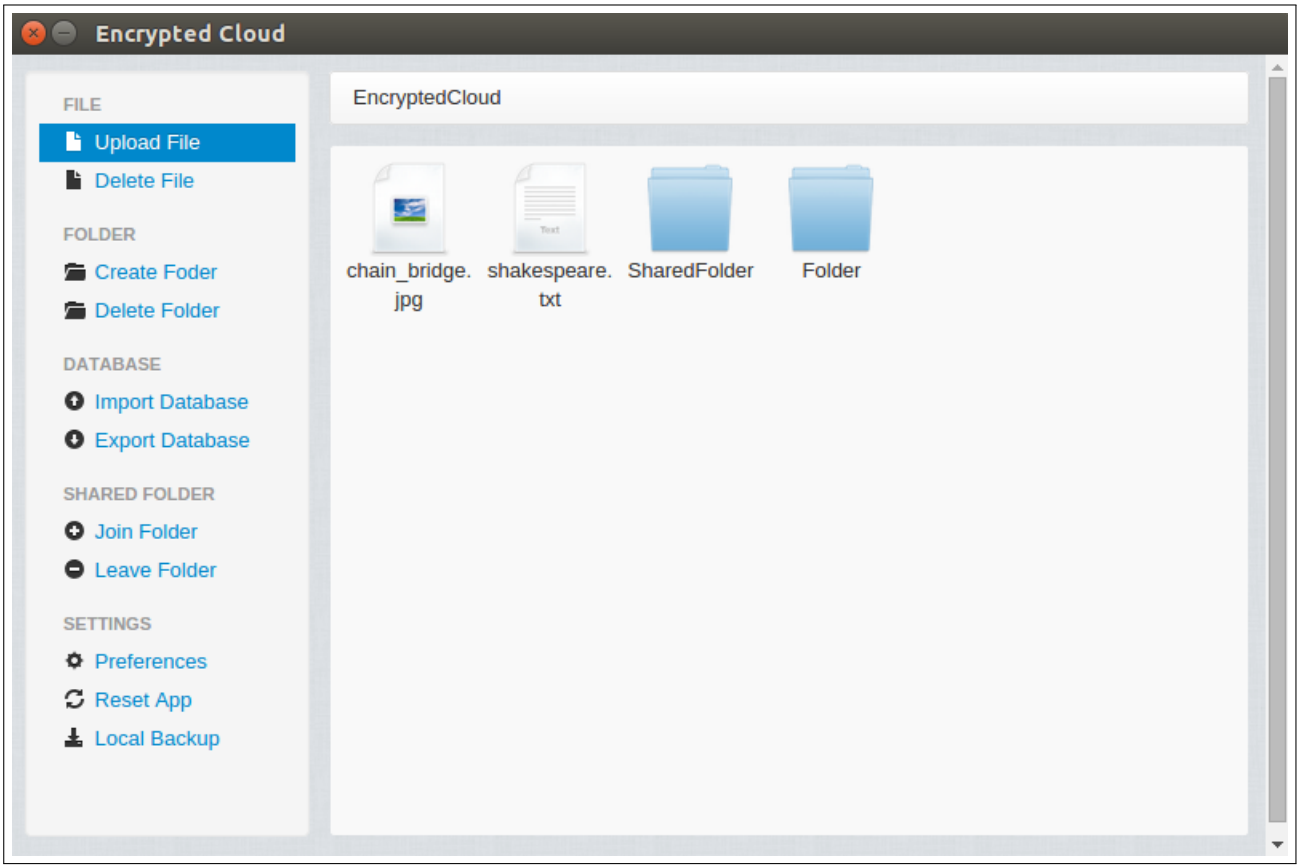


Figure 4.9: Encrypted Cloud file explorer.

4.5 Implementation

This section details all aspects related to the implementation of Encrypted Cloud.

4.5.1 Technologies used

Encrypted Cloud development was made with a new technology called NW.js[6], formerly known as node-webkit, which allows to create desktop applications on Windows, Linux and Mac OS X, being very simple the process of packaging and distributing these applications. With this technology, developers have a new way of programming native applications with the most popular Web technologies such as HTML5, CSS3, JavaScript and Node.js. Currently, there is a tendency to develop HTML5 applications in order to be functional on multiple platforms. Therefore, the development of Encrypted Cloud with this technology guarantees that it is portable to Windows,

[6] http://nwjs.io

Linux and Mac OS X environments. However, Encrypted Cloud has only been tested on Windows and Ubuntu due to not having any device with Mac OS X.

On the one hand, the view of Encrypted Cloud was developed using the popular Web technologies HTML5, CSS3 and JavaScript. All these technologies allow to move all the appearance and functionality that can be achieved in web pages to a desktop application. On the other hand, the logic of the application has been developed with Node.js, which is a server-side JavaScript environment based on events.

SQLite3 was used for the Encrypted Cloud database. Within this technology, it has been used a specific package called *sqlite-cipher*[7], developed by a member of the NPM community that allows to encrypt databases.

Finally, Encrypted Cloud has been implemented with NW.js 0.12.3 and Node.js 4.4.0. Within Node.js, Encrypted Cloud uses the eleven packages shown in Table 4.1.

Package	Version
chai	3.5.0
cron	1.1.0
fs-extra	0.26.7
jade	0.27.2
mocha	2.4.5
node-notifier	4.5.0
node-rsa	0.3.2
nodejs-fs-utils	1.0.26
random-seed	0.3.0
sqlite-cipher	0.3.4
underscore	1.3.3

Table 4.1: Node.js packages used in Encrypted Cloud.

4.5.2 Tools used

Along the writing of this document, the management of the references consulted was a really important aspect. For this task, it was really useful Mendeley[8], which is a free reference manager that works as a desktop program (for Windows, Mac and Linux) that allows to collect, share via Internet, manage and build collections of literature references.

On the other hand, the unit tests of Encrypted Cloud were performed using two tools. The first one is called *mocha*[9], which is a feature-rich JavaScript test framework

[7] https://www.npmjs.com/package/sqlite-cipher
[8] www.mendeley.com
[9] www.mochajs.org

running on Node.js and in the browser. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Along with *mocha*, *chai*[10] was used as assertion library.

4.5.3 Structure and code documentation

The project was divided into fourteen different libraries, in order to group each set of similar functionalities into different files:

- **address_bar_library.js:** it changes the address bar of the application when the user navigates between different folders.
- **async_tasks_library.js:** this library has three asynchronous tasks. The first one checks if all assets previously uploaded to Encrypted Cloud are actually in their path. The second task has the same objective as the previous one, but in this case with the requests sent to each shared folder. The third task deals with the updating of each shared folder. These three services run every thirty seconds.
- **crypto_library.js:** it has all cryptographic functions needed in Encrypted Cloud except the RSA key generation.
- **database_library.js:** it is the library that manages all operations with the encrypted database. For this, it uses the *sqlite-cipher* package.
- **folder_view_library.js:** it gets all files and folders that have to be displayed in the current view of Encrypted Cloud. The elements that are shown depend on which path the user is.
- **fs_library.js:** this is one of the core libraries of Encrypted Cloud, because it contains all functionalities related to file I/O operations.
- **global_variables_library.js:** it contains all the global variables used by the other libraries of Encrypted Cloud.
- **keygenerator.js:** it generates a 2048 bits RSA key pair in a child process, in order not to block the view of Encrpyted Cloud.
- **left_panel_actions_library.js:** this library manages all the operations that are available on the left panel on Encrypted Cloud, such as Upload File, Create Folder, Import Database, etc.

[10] www.chaijs.com

- **mime_library.js:** comprehensive MIME type mapping API based on the mime-db module, that allows to get different properties from files and folders.
- **modifyHTML_library.js:** as Encrypted Cloud is a single-page application, this library is needed for changing the content of the current HTML view.
- **notifier_library.js:** it is a module for sending cross-platform system notifications.
- **shared_folder_library.js:** this library is in charge of all the operations with shared folders: join/leave a shared folder, update a shared folder, etc.
- **start_app_library.js:** it manages each start of the application, in order to set or get all the configuration settings of Encrypted Cloud.

Regarding code documentation, note that each of the previous libraries has been documented in detail. This is an extremely important aspect, since it facilitates code maintenance and offers the possibility to reuse part of the program in other applications without having to know in depth the implemented code.

4.5.4 Implementation of cryptographic functionalities

This section details all the cryptographic functionalities used in the implementation of Encrypted Cloud with NW.js.

Taking into account the requirements specified in Section 4.3, it is necessary the use of both symmetric and asymmetric encryption. Since a mandatory rule in the information security world is avoiding the creation of cryptographic algorithms that are already implemented, in Encrypted Cloud it is only used cryptographic functions which are already developed in different Node.js packages.

4.5.4.1 Cryptographic libraries in Node.js

The great majority of the cryptographic functions of Encrypted Cloud has been implemented with the *crypto*[11] Node.js module. This package provides a cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign and verify functions. However, other Node.js packages have been used for specific functionalities which are detailed in the next subsections.

[11] www.nodejs.org/api/crypto.html

4.5.4.1.1 Symmetric encryption

Symmetric encryption is used to encrypt all the assets as uploaded to the cloud server, since this type of operation is carried out faster if we use symmetric encryption instead of asymmetric encryption. The algorithm chosen is AES 256, since it is the current standard for symmetric encryption. CBC was the operating mode selected for this algorithm, which is safer than the standard mode, known as ECB. Furthermore, AES 256 CBC has also been used to encrypt Encrypted Cloud database. Note that each symmetric key and initialisation vector is generated from a seed: the password that the user uses for authentication in Encrypted Cloud. This key is generated with the Node.js *random-seed*[12] package.

4.5.4.1.2 Asymmetric encryption

The well-known RSA asymmetric cipher is essential in Encrypted Cloud. It is used to sign all the files as they are uploaded to Encrypted Cloud, and to distribute a symmetric key among multiple users, functionality that is detailed in Section 4.5.4.2.

Finally, the generation of the RSA key pair is made with the *node-rsa*[13] Node.js package. The key size used in this algorithm is 2048 bits, as it is recommended in [98, p.66].

4.5.4.1.3 Digital signature

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or document. Each time a user uploads a file to Encrypted Cloud, it is digitally signed by this user, using again the *node-rsa* package. Then, when this file is downloaded, the user can verify if the signer is legitimate or not. In addition, if the file is modified by an unauthorised party, the user will be notified, but she will not be able to recover the original file since these files are not stored in the Encrypted Cloud database.

4.5.4.1.4 Hash functions

A hash function takes a group of characters and maps it to a value of a certain length (hash). The hash value is representative of the original string of characters, but

[12] www.npmjs.com/package/random-seed
[13] www.npmjs.com/package/node-rsa

is normally smaller than the original. In Encrypted Cloud, this type of cryptographic algorithm is used in the digital signature of each file uploaded to the application. The hash chosen was SHA-256 because for this algorithm, it has not been found collisions as it is shown in Figure 4.10.

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collisions found
MD5 (as reference)		128	128	512	$2^{64} - 1$	32	64	+,and,or,xor,rot	Yes
SHA-0		160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rot	Yes
SHA-1		160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rot	Theoretical attack (2^{60})
SHA-2	*SHA-224* *SHA-256*	224 256	256	512	$2^{64} - 1$	32	64	+,and,or,xor,shr,rot	None
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512	1024	$2^{128} - 1$	64	80	+,and,or,xor,shr,rot	None
SHA-3		224/256 /384/512	1600 (5×5 array of 64-bit words)			64	24	and,xor,not,rot	None

Figure 4.10: Comparison of SHA functions [99].

4.5.4.2 Symmetric key distribution

The distribution of a symmetric cryptographic key between multiple users is necessary in order to implement the functionality of sharing an encrypted file through a shared folder. Therefore, a protocol was needed for uploading an encrypted file to a shared folder and enabling its decryption by any user who belongs to this folder. As stated in Section 4.5.4.1.1, each file encryption is performed with a symmetric AES 256-bit key in CBC mode. In order to make the distribution of this key so that any user can decrypt the file, a packet was created as shown in Figure 4.11. The format of this packet is based on the digital envelope, a container with a message encrypted with a symmetric encryption, also including in this container the symmetric key. Since this key must be protected, it is encrypted using the receiver's public key.

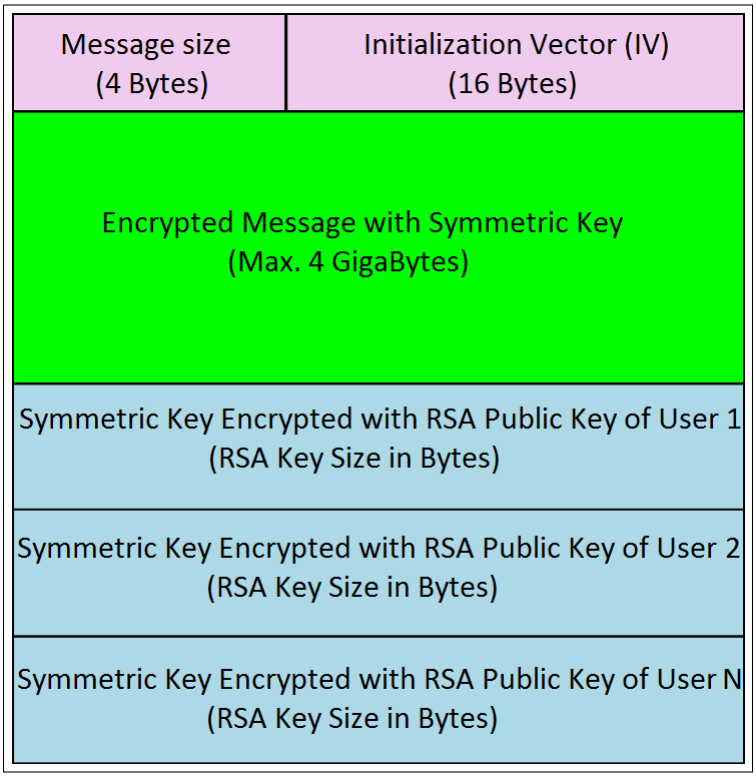


Figure 4.11: Key distribution packet.

The first field of the packet has a size of 4 Bytes, and indicates the size of the encrypted message contained in the packet. Therefore, this size can be a maximum of 4 GigaBytes.

The next field has a size of 16 Bytes and represents the initialisation vector needed in AES 256 CBC.

Then, the following field is the message encrypted with AES 256 CBC.

After the encrypted message, the symmetric key is encrypted with the RSA public key of each user with whom the user wants to share the message and who belongs to the shared folder. The order of this symmetric key encryption is equal to the order in which users are joined to the shared folder, stored in a special file which will be explained in Section 4.6.1.1.1. The size of these fields is the same than the RSA key size in bytes.

Finally, the last component of the packet is the digital signature of the packet itself. This signature is computed by the user who uploads the file to a shared folder. This signature has the same size as the RSA key in bytes.

Therefore, when a user wants to decrypt a packet like this, the first thing she has to do is to know her position within the shared folder in order to access the symmetric key encrypted with her RSA public key. After that, she will be able to obtain the

symmetric key. As a result, with this key along with the initialisation vector, she will be able to get the original plain text message.

4.6 Encrypted Cloud functionalities

This section describes in detail how the developed application works. First, some initial considerations are required for the sake of the better understanding of certain features of the application. Then, Section 4.6.2 presents each functionality of Encrypted Cloud.

4.6.1 Considerations to take into account

In this section there are some particular considerations of Encrypted Cloud. Additionally, some main obstacles found during the development of this tool are discussed.

4.6.1.1 Shared folders in Encrypted Cloud

There are different considerations related to shared folders that must be understood. First, this type of folders must be selected by the user in the Encrypted Cloud set up. This is necessary for later distinguish between shared and non-shared folders. Therefore, in this set up stage, the user should select these local folders which are already shared with other users, like Dropbox or Google Drive shared folders or network drives. However, Encrypted Cloud understands that these folders do not have any user joined to them, because it has its own protocol for all operations related to this kind of folders: join/leave a shared folder, upload/view files in a shared folder and create/delete folders into a shared folder. Moreover, these shared folders can be or not an inner folder of a non-shared folder. Finally, it is also important to know that an inner shared folder is shared by the same users joined to its root shared folder.

4.6.1.1.1 Users management in shared folder

Encrypted Cloud has its own protocol to manage both join and leave actions in shared folders. First, remember that when it is the first time that the user starts the application, a RSA key pair is created. This key pair is always generated by the user in the client application.

It is important to recall that Encrypted Cloud interpreted that a shared folder between two users is an existing connection between them. Nonetheless, the sharing of files through Encrypted Cloud can be only carried out if those users are joined to this folder. The implemented protocol establishes that a shared folder has a file

called "*.usersFolder.txt*" whose content is a list of users joined to that shared folder. Each user is represented by her RSA public key, that is separated from the rest of RSA public keys by the character ';' (see Figure 4.12).

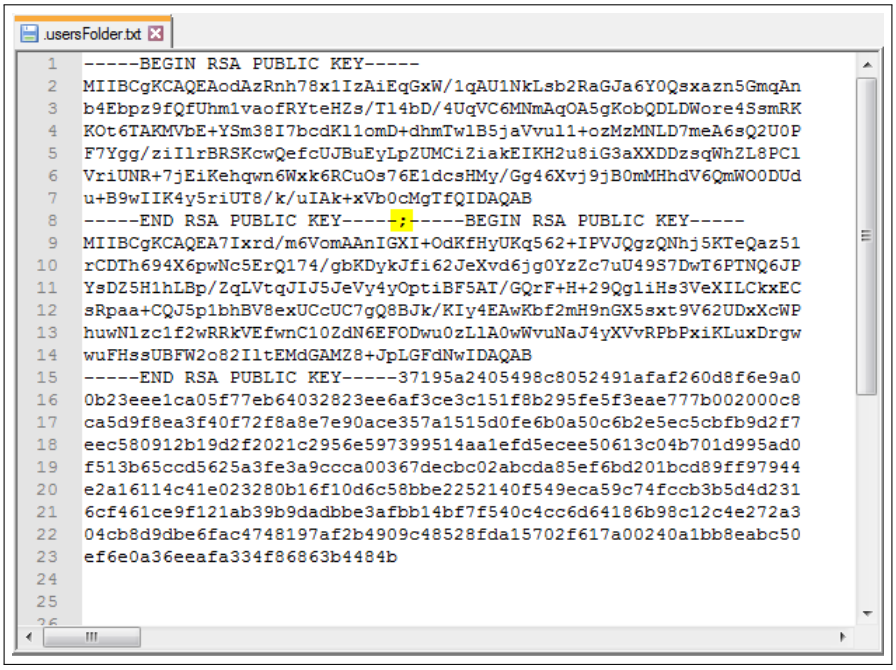


Figure 4.12: Format of "*.usersFolder.txt*".

The first user joined to a shared folder will be the owner and her RSA public key will be the first in the user file. The "*.usersFolder.txt*" file is always digitally signed by the owner of the shared folder. Other users who want to join or leave a shared folder must send a request to the owner of the folder. This behaviour is described in Sections 4.6.2.3.1, 4.6.2.3.2, and 4.6.2.3.3.

4.6.1.2 Obstacles found in the implementation

We have found different obstacles during the implementation of Encrypted Cloud. Therefore, it was necessary to apply some workarounds in order to solve them.

The RSA key generation was initially made with other Node.js package called *ursa*[14], which is based on OpenSSL. This package works fine on Ubuntu, but it has several installation problems on Windows. Thus, this was the reason why it was finally used *node-rsa*, because this library does not need OpenSSL since it is based on jsbn library[15]. With this type of workaround, it is assured that Encrypted Cloud does not require the installation of other external applications.

[14] www.npmjs.com/package/ursa
[15] www-cs-students.stanford.edu/~tjw/jsbn

Moreover, it was also a challenge the identification of hidden files and folders in both Ubuntu and Windows platforms. In Ubuntu, it is easy to make this distinction, because all the names of hidden elements begin with a dot. But it is not the same on Windows, since its hidden elements have a special property which does not depend on the dot at the beginning of their names. In order to solve this issue, first it was used a Node.js package called *hidefile*[16], but it did not work fine with NW.js in Windows, because this package, like Node.js, is Unix-first (there is an open issue in GitHub to solve this limitation). Therefore, a solution for this problem was not found, because another possible workaround would be to make call systems from Encrypted Cloud, but this approach will make the application OS dependent.

4.6.2 Functionalities

This section explains step by step each functionality of Encrypted Cloud, so it can be used as a user guide.

4.6.2.1 Encrypted Cloud set up

The screen that appears when the application is started for the first time is shown in Figure 4.13.

[16] www.npmjs.com/package/hidefile

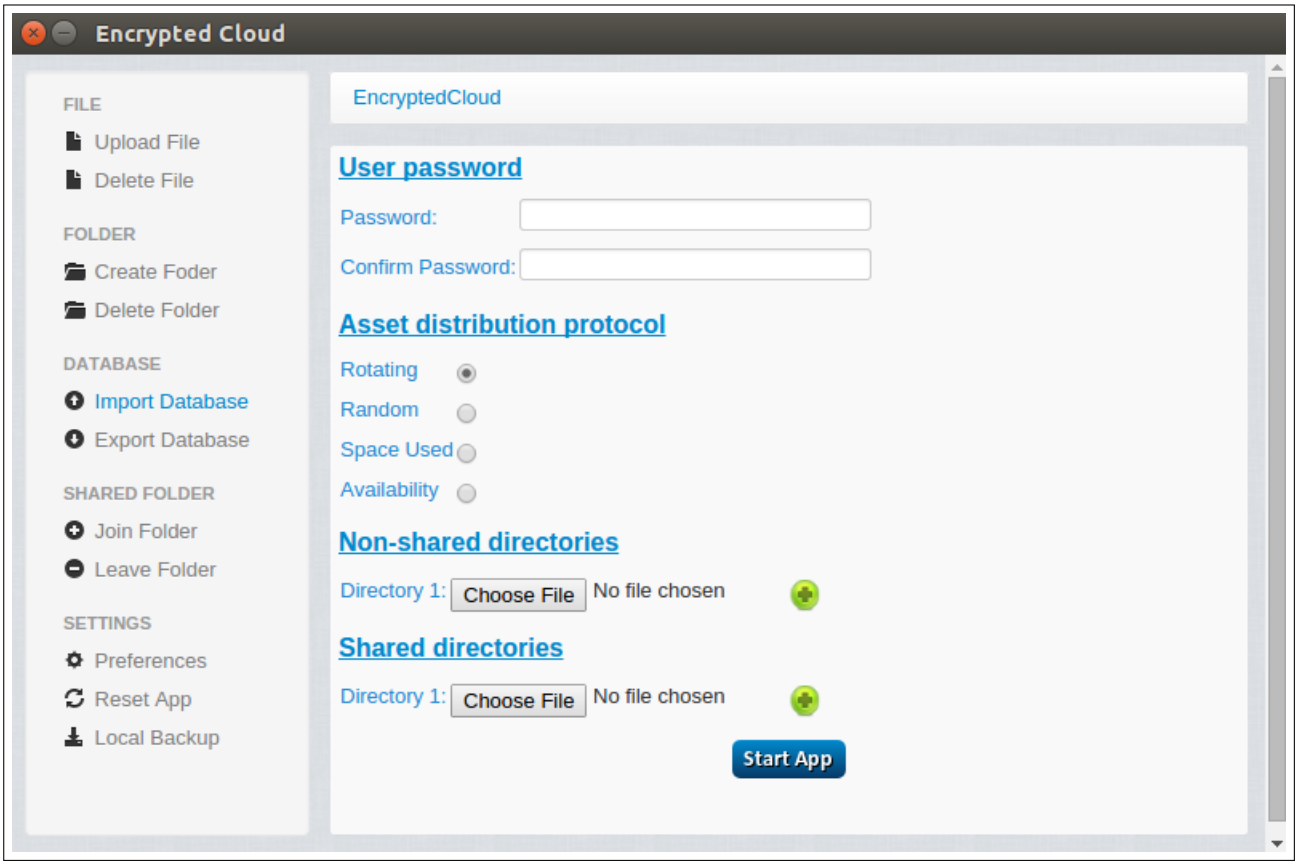


Figure 4.13: Set up.

The main form has four different sections:

- **User password:** the user has to type her password twice to be registered and subsequently establish the password-based authentication process. The provided password is also used as a seed to generate the symmetric key that encrypts the non-shared files and the local database.
- **Asset distribution protocol:** the user must select which asset distribution protocol wants to use. This protocol applies to both the file upload and creation of new folders, and it will determine in which non-shared directories will be stored each asset.
- **Non-shared directories:** the user must select at least one non-shared directory where deposit those assets that cannot be shared with other users.
- **Shared directories:** the selection of shared directories is optional. If the user wants to share assets with other users, then she must select at least one shared directory to store shared elements.

Finally, when the user clicks on the "Start App" button, each field on the form is verified. If there is no error, it will navigate to the file explorer of Encrypted Cloud.

4.6.2.2 Encrypted Cloud login

When the user starts the application, the database could be already created, which means that the set up was previously made. In this case, the user must authenticate herself by providing her password (see Figure 4.14).

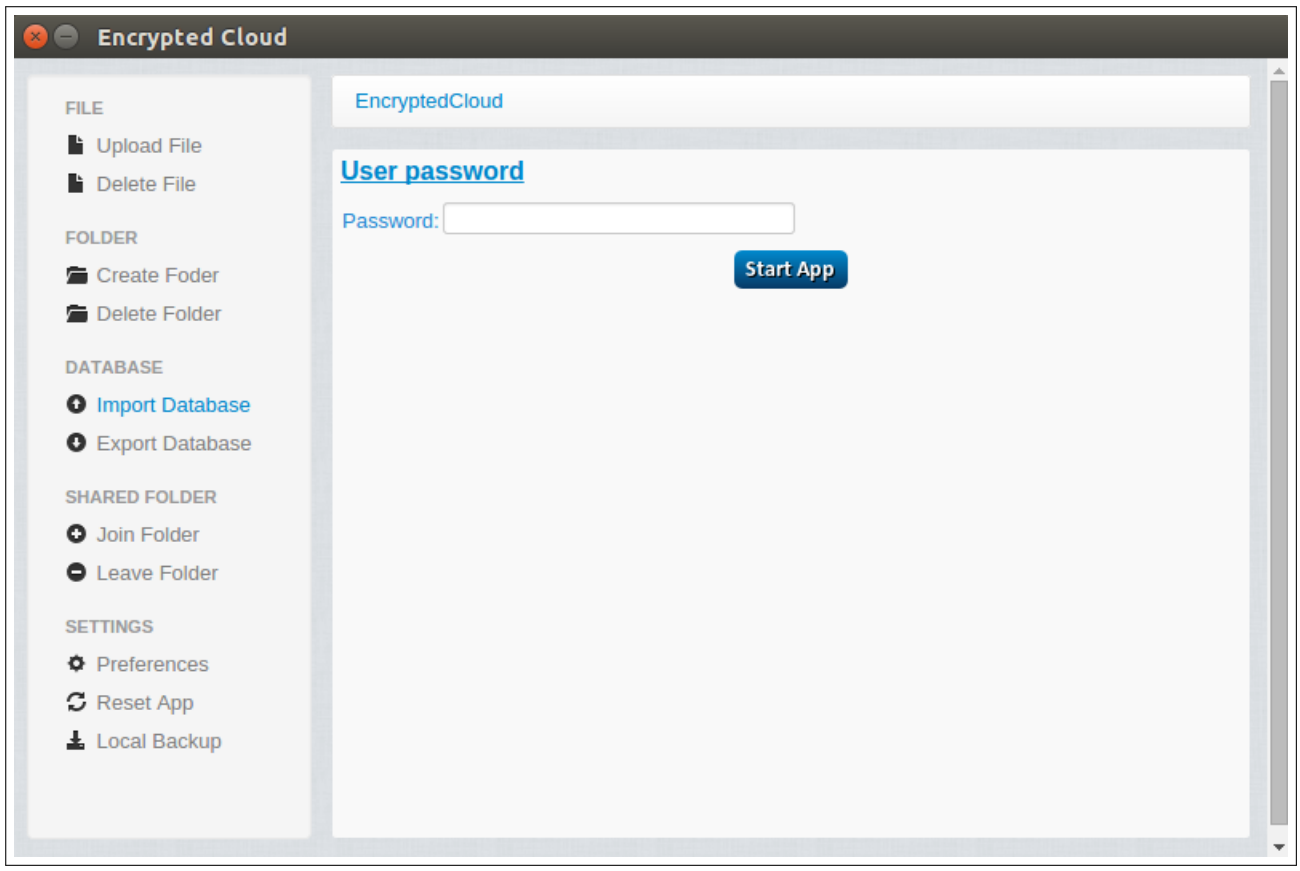


Figure 4.14: Login.

The authentication process tries to connect to the database using the password provided by the user. If this process throws an error, it means that the user's password is incorrect or the database is corrupted. In the latter case, the user should import a previous backup of the database through the left panel option "Import Database". On the other hand, if the authentication process is successful, the application will redirect the user to the file explorer.

4.6.2.3 Encrypted Cloud file explorer

Once the user has made the set up of the application or has logged in correctly, she will see the main screen of the application (see Figure 4.15).

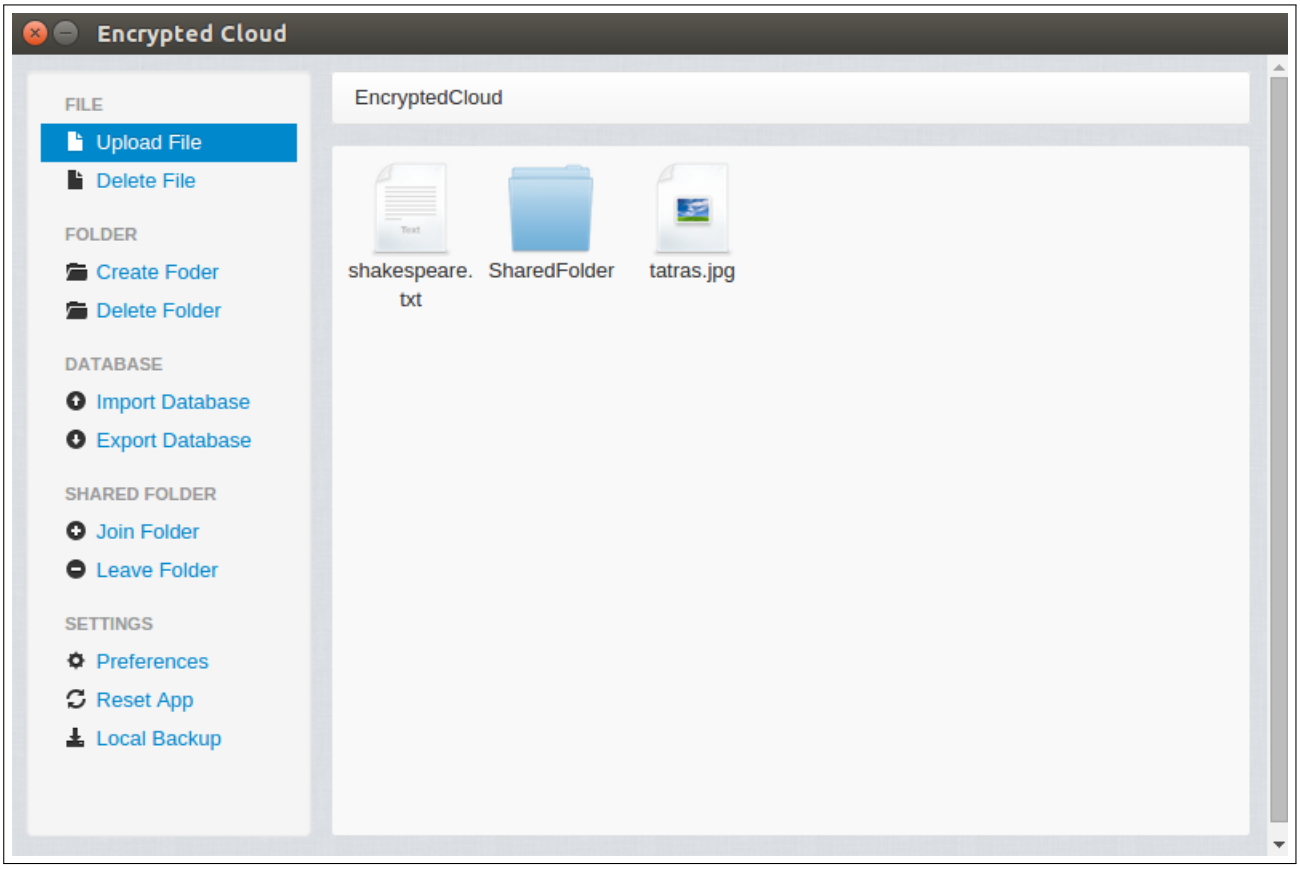


Figure 4.15: File explorer.

The appearance of this screen is very similar to any other file browser, but it does not allow neither multi-element selection nor right-click menu. On the left side panel there are different actions that will be described in subsequent subsections. The central part displays all files and folders (non-hidden) of both shared and non-shared folders. Say that at the root path of this file explorer, all the elements contained in the non-shared folders are shown along with each of the shared folders selected. Thus, the user sees the non-shared folders as if they were a single directory.

4.6.2.3.1 Join a shared folder

The functionalities of join and leave a shared folder are explained before than upload and view files because the latter depend on the former.

In order to join a shared folder, the user first should select the shared folder and click on the link "Join Folder" shown in Figure 4.16.

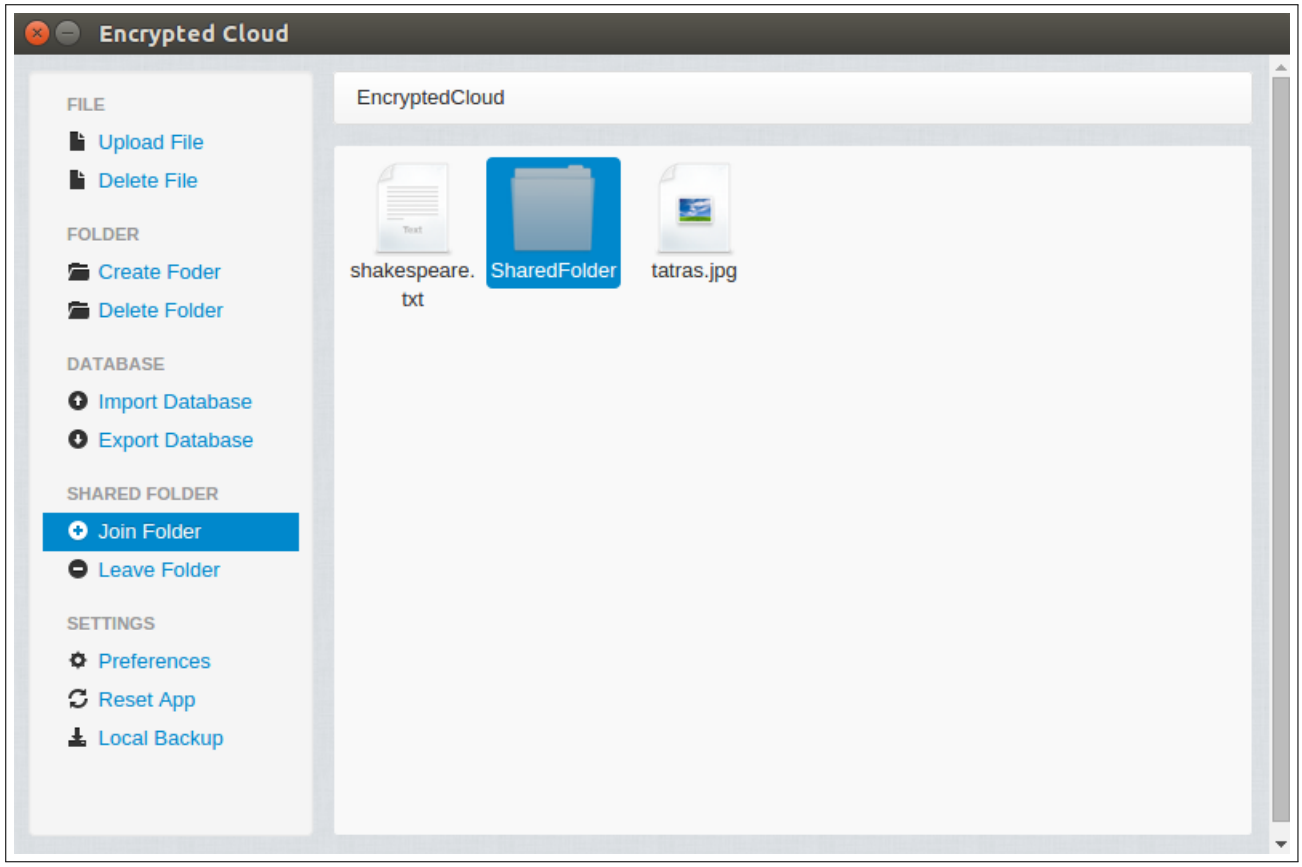


Figure 4.16: Join shared folder.

When the user chooses the folder, it is first checked if that folder has been selected as shared on the set up of the application. In addition, remember that all internal folders to a shared folder are shared by the users joined to the root folder. In this joining process, there are two different scenarios:

- If the user who wants to join the shared folder is the first that will be in that folder, then she will be the owner. This action will create the *.usersFolder.txt* file in the shared folder, adding her RSA public, and it will be uploaded digitally signed. After this process, the user will be notified that she has successfully joined to the shared folder as owner, as shown in Figure 4.17.

Figure 4.17: Notification of user joined to shared folder as owner.

- Otherwise, if there are other users joined to this shared folder, the user would have to send a join request to the owner of the shared folder. This request consists of a file whose name is the concatenation of the *appID* and an identifier (*_join.txt*, to distinguish if it is a join or leave request), and whose content will be her RSA public key digitally signed. This request will be uploaded to a folder within the shared folder that she wants to join, called *.join_requests*. Furthermore, this request is stored in the local database, specifically in the SharedFolderRequest entity. A malicious server could eliminate such requests, and consequently there would not be any proof about the requests for joining a shared folder. To tackle this matter, it was implemented a service that is in charge of monitoring if a request has been managed by the owner. This service checks every 30 seconds if all join requests stored in the SharedFolderRequest entity have been managed correctly, that is, verifying if the user is already included in the corresponding shared folder, checking if her public key is in the *.usersFolder.txt* file. In case it is not contained yet, the join request will be uploaded again. Once the user has successfully requested to join the shared folder, she will be notified as shown in Figure 4.18.

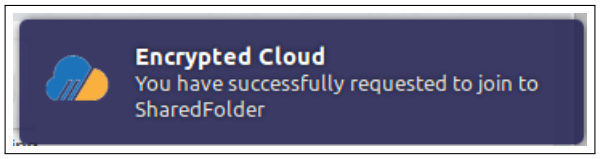


Figure 4.18: Notification of join request sent.

Finally, if the user tries to join twice to a shared folder or when it is being updated, an error will be shown.

4.6.2.3.2 Leave a shared folder

In order to leave a shared folder, the user first should select the shared folder and click on the link "Leave Folder" shown in Figure 4.19.

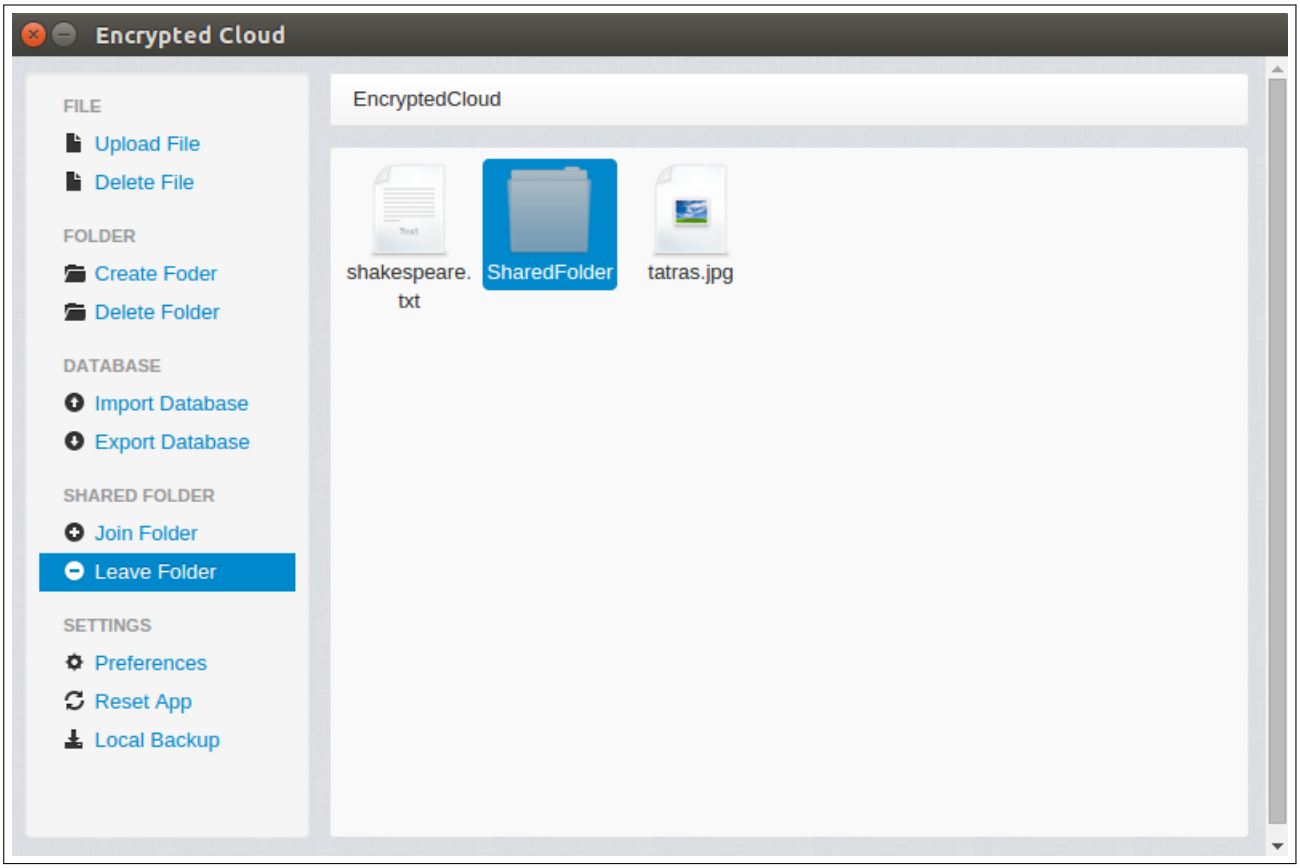


Figure 4.19: Leave shared folder.

When the user chooses the folder, it is first checked if that folder has been selected as shared on the set up of the application. In this leaving process, there are two different scenarios:

- If the user who wants to leave the shared folder is the owner, that is, her public key RSA is the first in the *.usersFolder.txt* file (because it was the first to join it), she only will remove all items contained in that folder, as it has taken the approach that when an owner leaves a shared folder, that folder is reset in order to use it again in the future. All registers stored for this shared folder in the Asset entity will also be removed. After this process, the user will be notified that she has successfully left the shared folder, as shown in Figure 4.20.

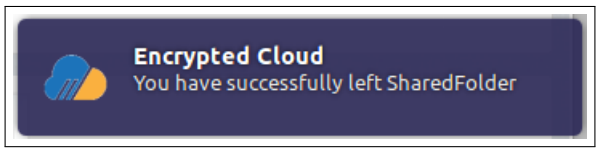


Figure 4.20: Notification of user left shared folder as owner.

- Otherwise, if the user is not the owner of the shared folder because her RSA public key is not the first in the *.usersFolder.txt* file, she would have to send a leave request to the owner of that shared folder. This request consists of a file whose name is the concatenation of the *appID* and an identifier (*_leave.txt*, to distinguish if it is a join or leave request), and whose content will be her RSA public key digitally signed. This request will be uploaded to a folder within the shared folder that she wants to leave, called *.leave_requests*. Furthermore, this request is stored in the local database, specifically in the SharedFolderRequest entity. Moreover, a malicious server could eliminate such requests, and therefore would not be a trace that a user has tried to leave the shared folder. To resolve this problem, there exists a service for verifying if the request has been managed by the owner. This service checks every 30 seconds if all leave requests stored in the SharedFolderRequest entity have been managed correctly, that is, it confirms if the user is not included in the corresponding shared folder, just by checking if her public key is not in the *.usersFolder.txt* file. In case it is contained, the leave request will be uploaded again. Once the user has successfully requested to leave the shared folder, she will be notified as shown in Figure 4.21.

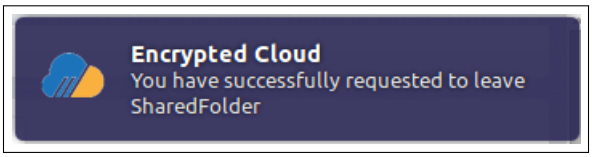


Figure 4.21: Notification of leave request sent.

Finally, if the user tries to leave twice a shared folder or when it is being updated, an error will be shown.

4.6.2.3.3 Update a shared folder

Each time that the user reaches the file explorer screen, it is executed a crucial asynchronous task for the application. This service is responsible for going through all shared folders selected in the application set up. For each shared folder, it checks if the current user is the owner. If so, it first verifies the integrity of the *.usersFolder.txt* file, in order to get the latest version of this file from the database in case of file has been deleted or modified. The next step is to see if that folder contains join and/or leave requests sent by other users.

As it has already been mentioned before, the content of these requests is the RSA public key digitally signed by the user who made the request. Therefore, the first

thing the owner has to do is to process each request contained in the shared folder, checking that the digital signature is correct. If it is not correct, the request will be deleted.

Once this verification process has been completed, the next step is to get the current content of the *.usersFolder.txt* file in the shared folder, in order to update it by adding and removing users, depending on the request types processed. We should note here that the *.usersFolder.txt* file contains all public keys of the users joined to the shared folder separated by ";".

Finally, when the *.usersFolder.txt* file is updated, the owner downloads each file of this shared folder, and decrypts and re-encrypts it with a new key AES 256, taking into account the new users contained in the shared folder. Then, each file is stored again in the shared folder. This process ensures a good management of the revocation and the joining of users, since the revoked users cannot access the content of the files in the shared folder, and new users have full access to them once it has completed the process of updating the folder. Furthermore, if the owner checks that a file has been improperly modified, it will be automatically deleted. During this process, the SharedFolderUsers entity is also updated with the new users joined to the shared folder.

This way of carrying out the management of the requests has the problem that when a user makes a request to an owner of a shared folder, she has to wait for the management made by this owner. In fact, in the current version of Encrypted Cloud we are implicitly trusting the CP when dealing with the public keys of the users. As it is implemented now, the CP could fake requests with her public key and there is no way to detect such an attack. As future solution, Encrypted Cloud will incorporate a new communication protocol (for example, via email) to import the public keys of the different users.

4.6.2.3.4 Upload a file

When a user wants to upload a file, she must go to the directory where she wants to place this file. Then, the user has to click on the "Upload File" link on the left side panel, which is highlighted in Figure 4.22.

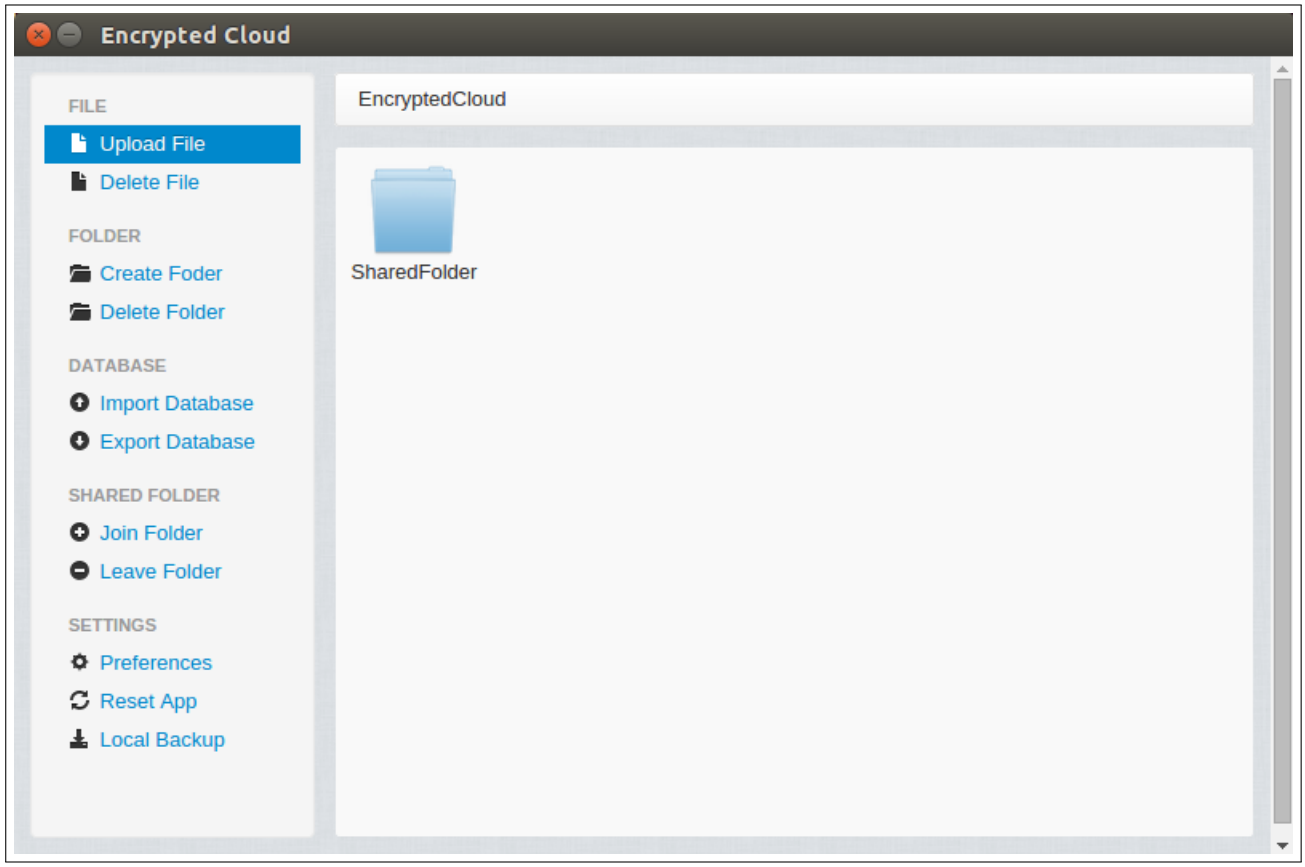


Figure 4.22: Upload a file.

After that, the user has to select the file she wants to upload through a file browser. She could select more than one file, as shown in Figure 4.23.

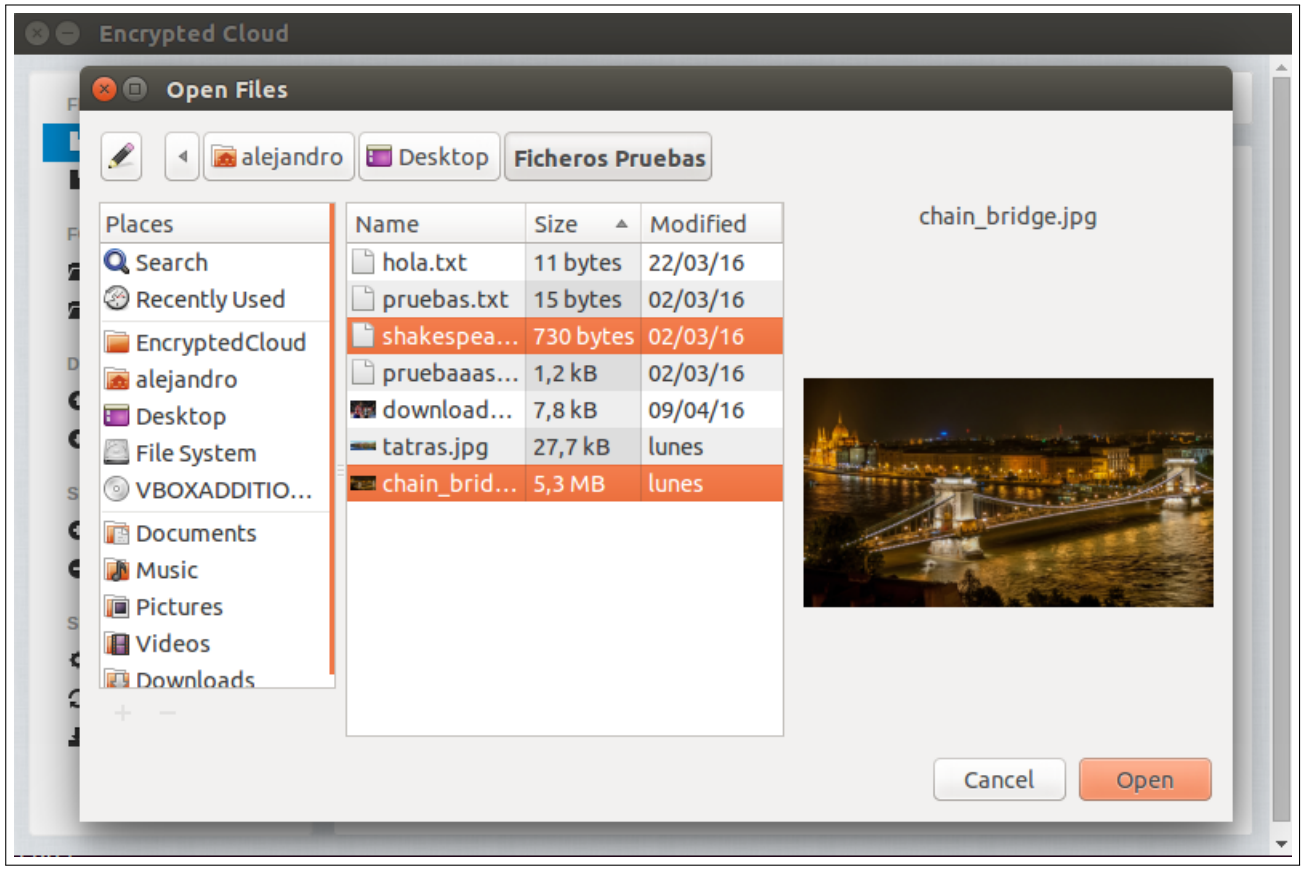


Figure 4.23: Multiple file selection from the file explorer.

Then, the next step is to prepare the file before uploading it to the destination folder:

- If the user is inside a non-shared folder, the selected file will be encrypted with AES 256 CBC, using as seed the user's password for the key generation. After this encryption, the file is digitally signed and uploaded to the current directory, taking into account the asset distribution protocol selected on the set up of the application. For example, if the current asset distribution protocol is "Availability", a copy of this file will be made in all non-shared directories. In addition, the uploaded file is stored in the Asset entity of the local database.

- On the other hand, if the user is inside a shared folder, she has to be joined to this shared folder and this folder cannot be involved into the updating process at this moment. Afterwards, the file is encrypted using AES 256 CBC, according to the packet format that it was explained in Section 4.5.4.2, which is responsible for distributing the encryption key of the file with the users of the shared folder.

Once it is formed the packet, it is saved in a file, and it is digitally signed and stored in the current shared folder. This file is only stored in the local database if the current user is the owner of the shared folder.

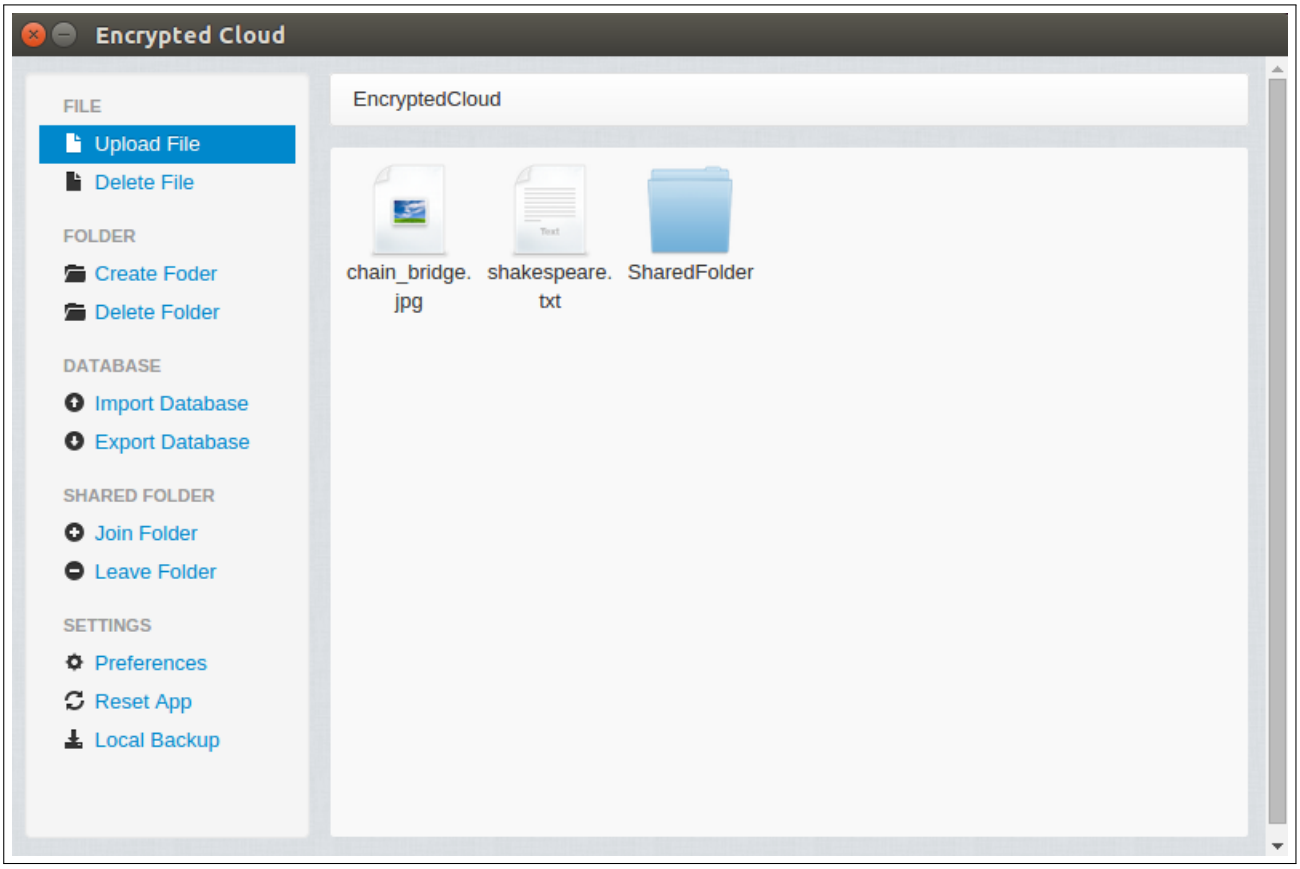


Figure 4.24: Files uploaded.

Moreover, if the user tries to view the file from outside Encrypted Cloud, she could not display it (see Figure 4.25).

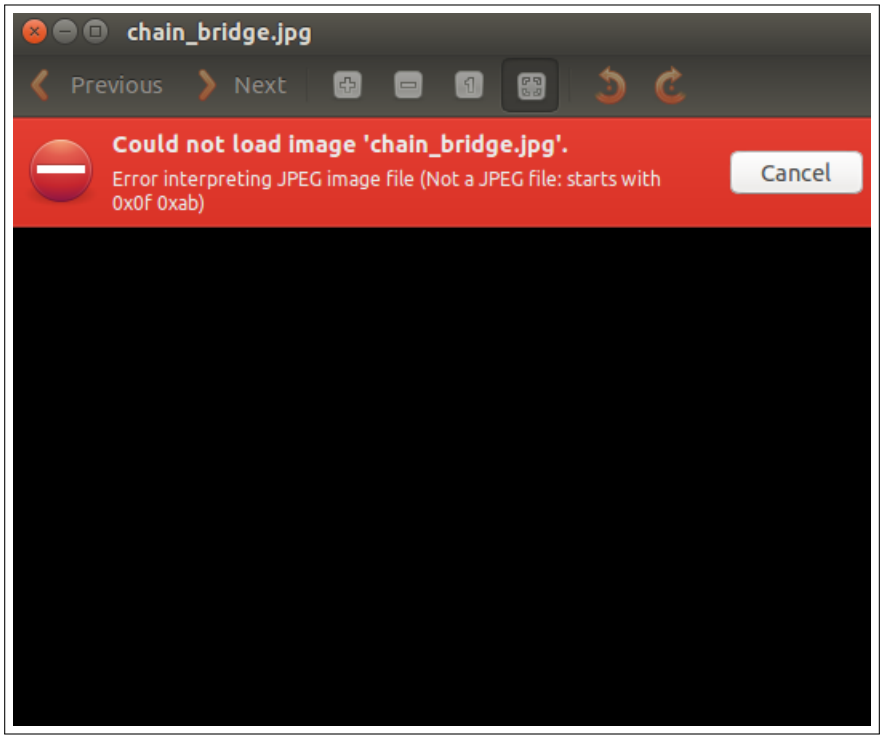


Figure 4.25: Error opening encrypted file.

It is also supported a drag and drop functionality for uploading a file through the application.

4.6.2.3.5 View a file

If the user makes a double click on a file, it will be opened with the appropriate external application. Before she can open this file, it is checked if this file is contained in a shared folder, and in this case, it only can be visualised if the current user is joined to that shared folder. Afterwards, the file is processed in order to be displayed:

- If the file is contained in a non-shared folder, first it is verified that it is correctly digitally signed. Then, the AES 256 key is generated with the seed obtained from the user's password in order to perform the decryption.

- Otherwise, if it is a shared file, first it is checked that the folder is not being updated at the moment. After that, the digital signature verification is made, which involves to contrast that the signature of this file has been made by one of the users joined to the shared folder. The next step is to get the symmetric key of this encrypted packet applying the mechanism explained in Section 4.5.4.2. Once the symmetric key is obtained, the user will be able to decrypt the file.

Whenever the digital signature of a file is incorrect, the user will be notified about this problem. These corrupted files are deleted only in the case that they are shared and when they are processed by the owner of the shared folder.

Finally, when the file has been decrypted, it is first stored in a local folder called *tmpFiles* and then displayed to the user through the most appropriate external application, as shown in Figure 4.26.

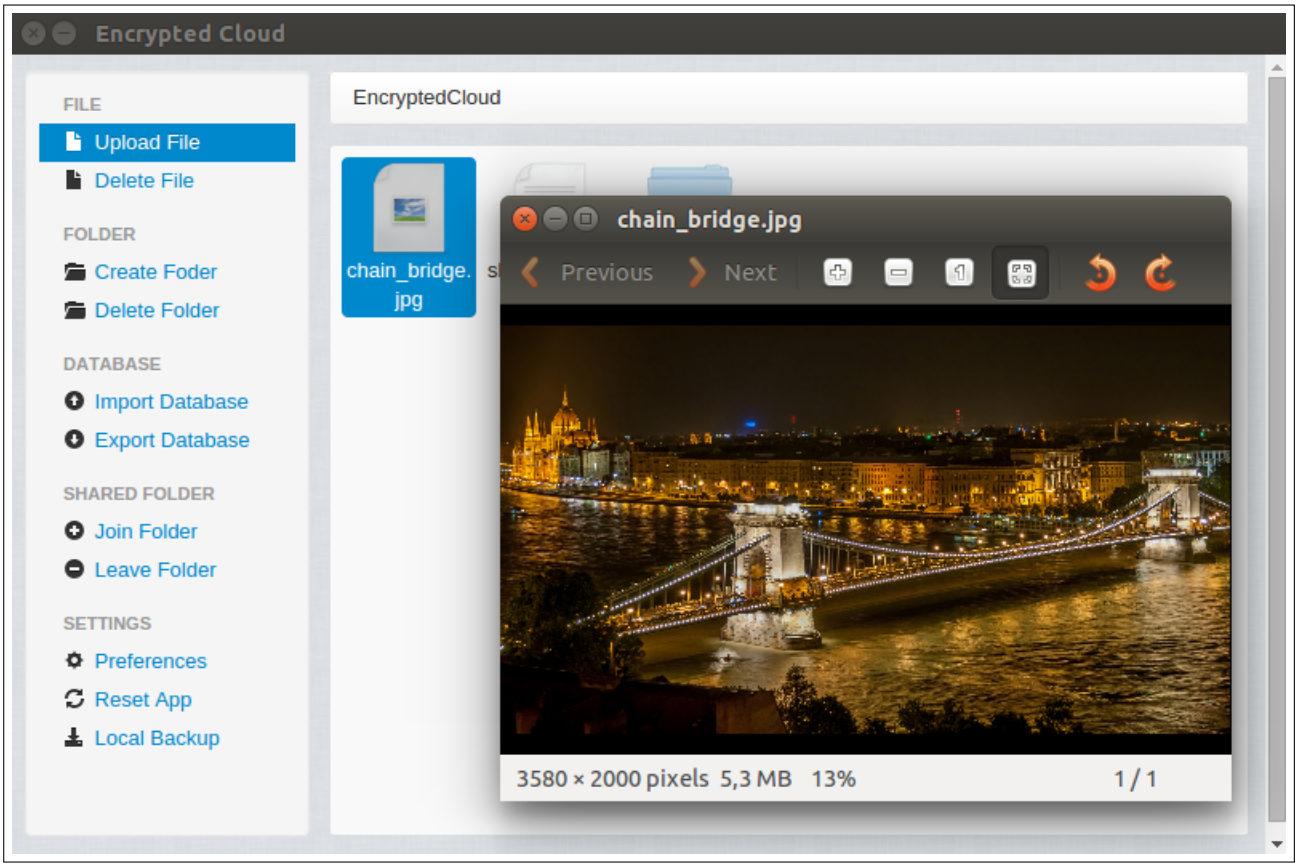


Figure 4.26: View Encrypted Cloud file.

All files contained in *tmpFiles* folder are deleted when the user closes the application.

4.6.2.3.6 Delete file

The user can delete a file previously uploaded to Encrypted Cloud. She only has to select the file that she wants to remove and click on "Delete File" link on the left panel, as shown in Figure 4.27.

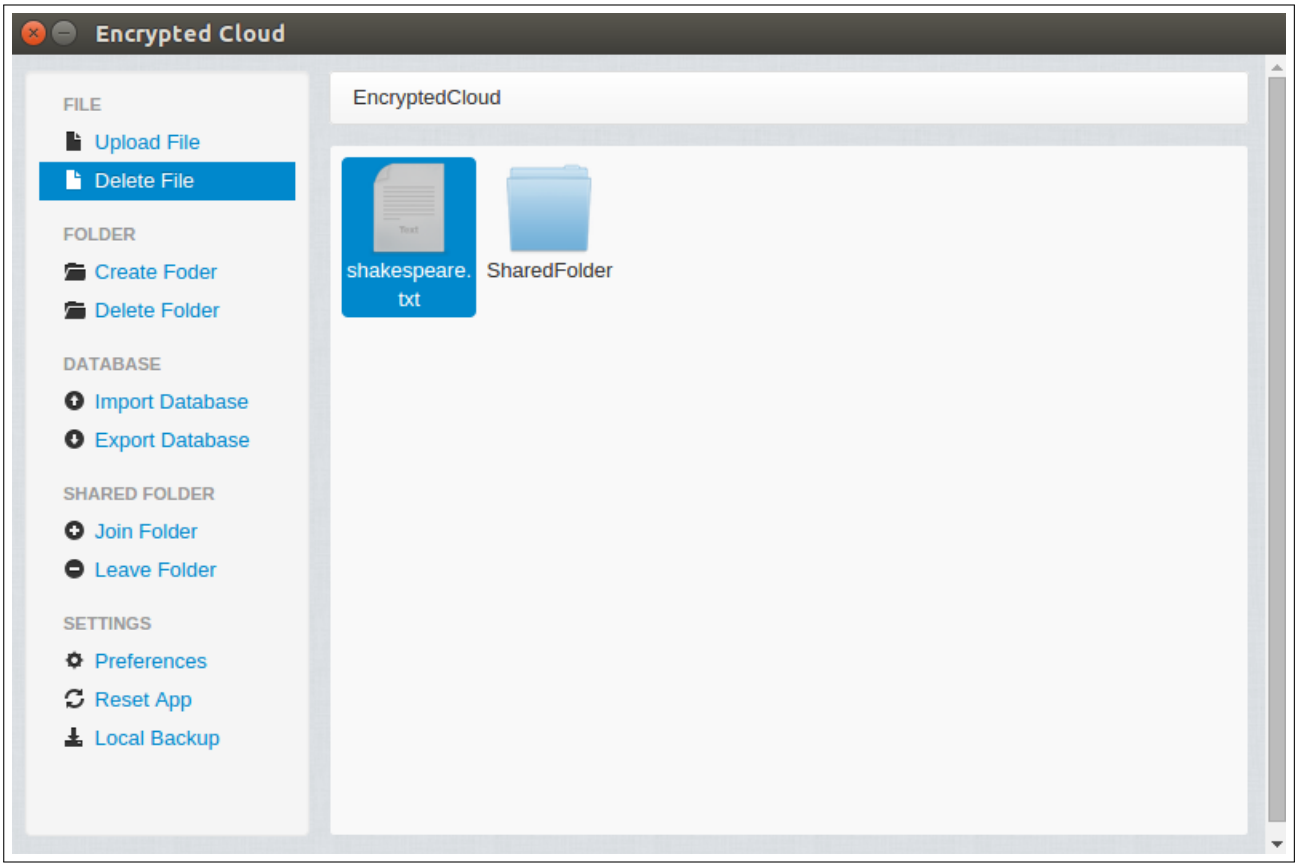


Figure 4.27: Delete Encrypted Cloud file.

She can delete all non-shared files and only these shared files contained in shared folders to which the user is joined. Additionally, if the current asset distribution protocol is "Availability", all copies of the file selected will be removed. It is also deleted this file from the Asset entity of the local database.

4.6.2.3.7 Create folder

Encrypted Cloud also provides the option to create folders. The procedure is really similar to the file upload. First, the user has to be in the path where she wants to create the new folder. The next step is to click on "Create Folder" link on the left panel and type the folder name in the following pop-up (Figure 4.28).

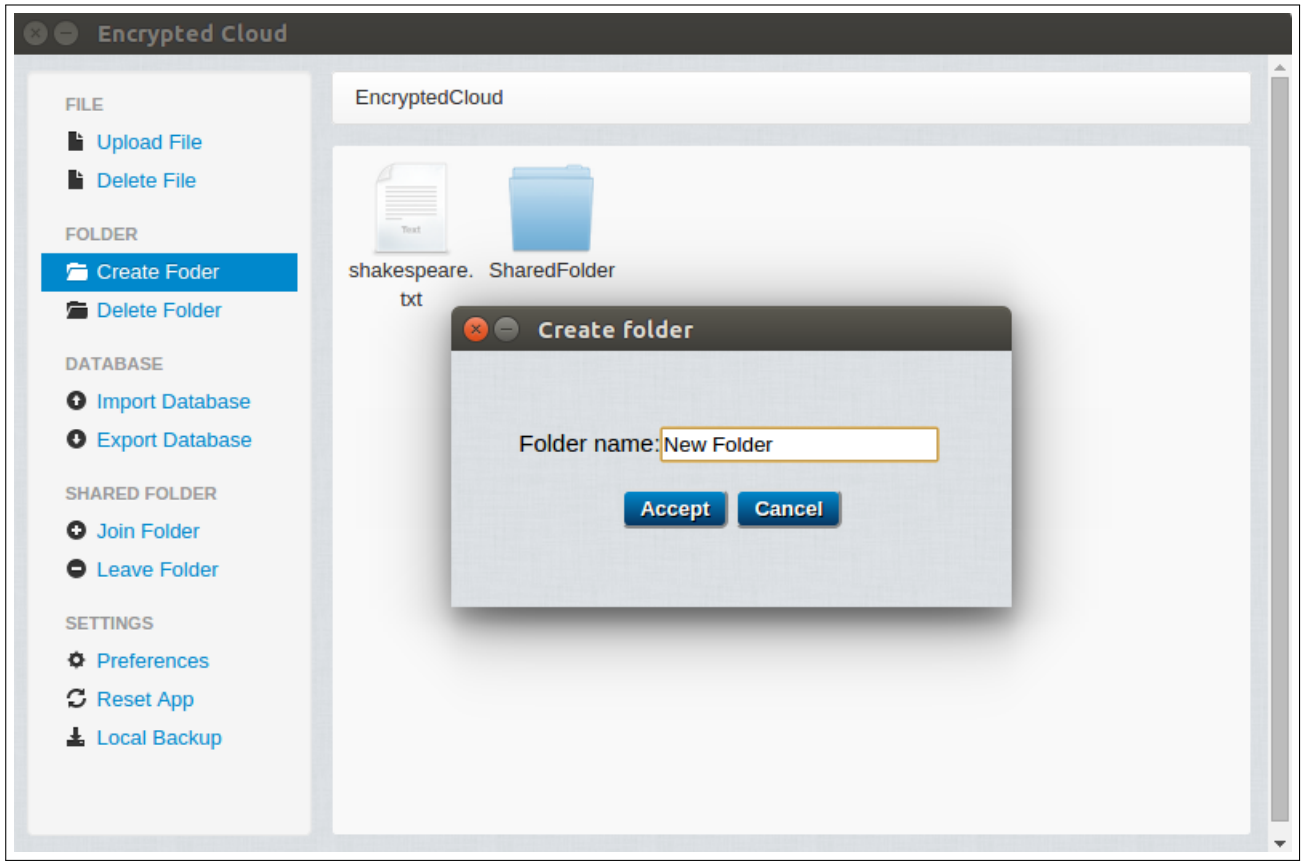


Figure 4.28: Create new folder.

If the current path of the user is a shared folder, it is verified that the user is joined to the shared folder. After that, it is also checked that it does not already exist other folder with the same name. Then, the folder could be created, keeping in mind the asset distribution protocol and saving this folder path in the Asset entity.

4.6.2.3.8 Delete folder

This functionality is almost the same as delete a file. The only difference is that it is not possible to delete a shared folder who was selected in the set up of Encrypted Cloud. Therefore, it is only allowed to remove the contents of the "root shared folders" (both files and folders). To carry out this action, the user needs to select the folder that she wants to remove and click on "Delete Folder" link on the left panel:

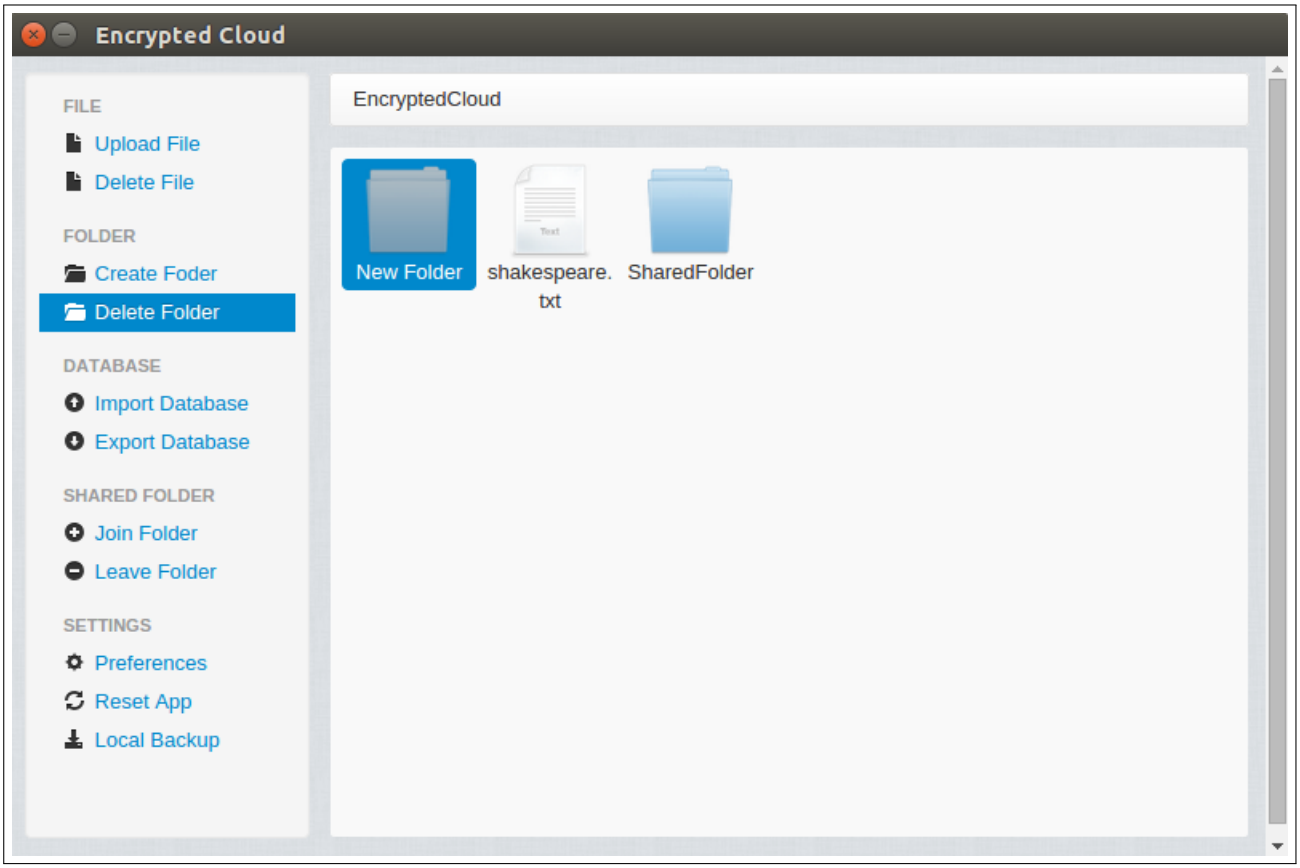


Figure 4.29: Delete Encrypted Cloud folder.

When a folder is selected to be deleted, its contents are also removed.

4.6.2.3.9 Import database

Encrypted Cloud provides the functionality of importing a local database. For this, the user has to click on the "Import Database" link on the left panel. Next, the user has to select the database file that she wants to import through a file explorer. When this action is accomplished, it is shown a pop-up that confirms the operation:

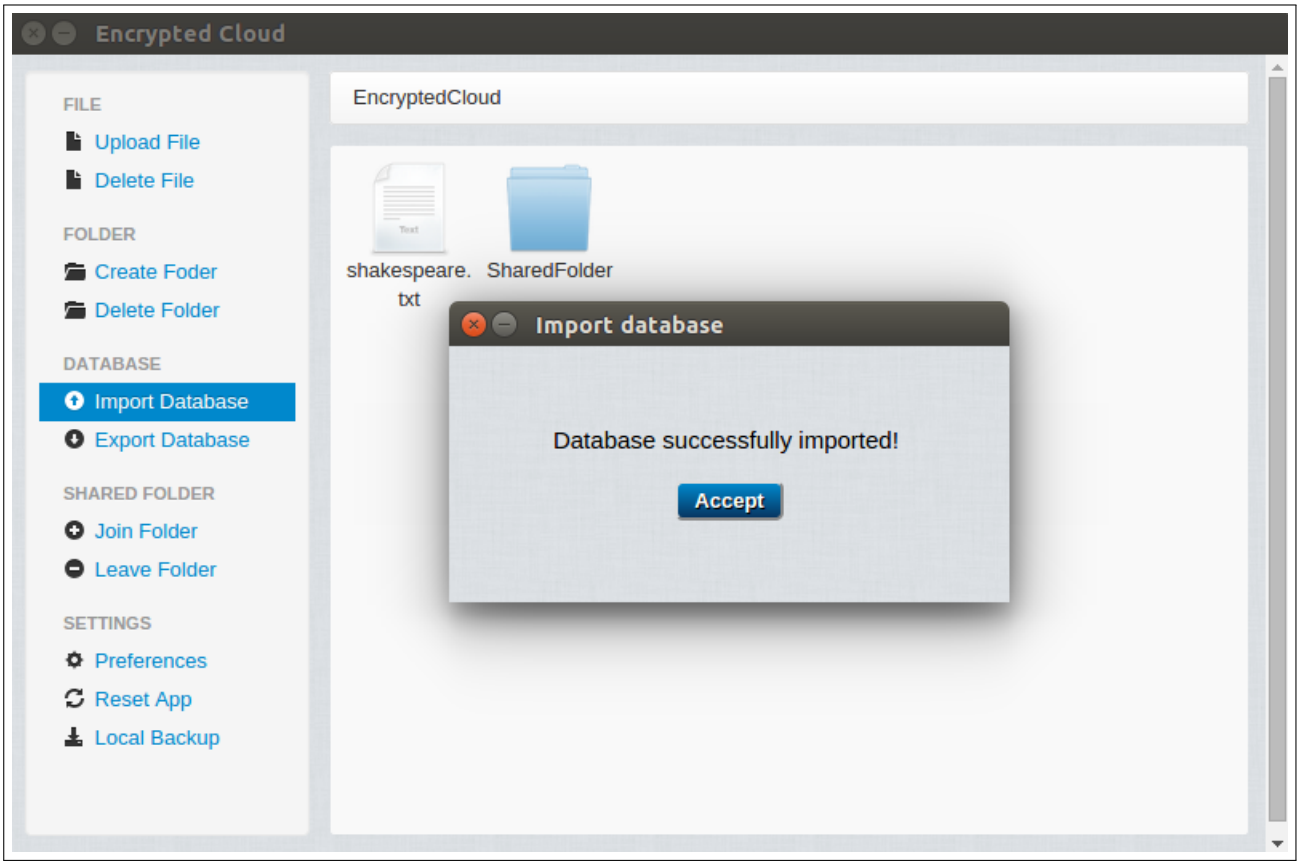


Figure 4.30: Import database pop-up.

This functionality is very important if she wants to recover a previous state of the application or if the current database is corrupted.

4.6.2.3.10 Export database

Encrypted Cloud also provides the functionality of exporting a local database with the aim of saving the current application state in order to recover if it would be necessary. The user only has to click on the "Export Database" link on the left panel and select the destination folder where she wants to store this backup. When this action is finished, it is shown a pop-up that confirms the operation:

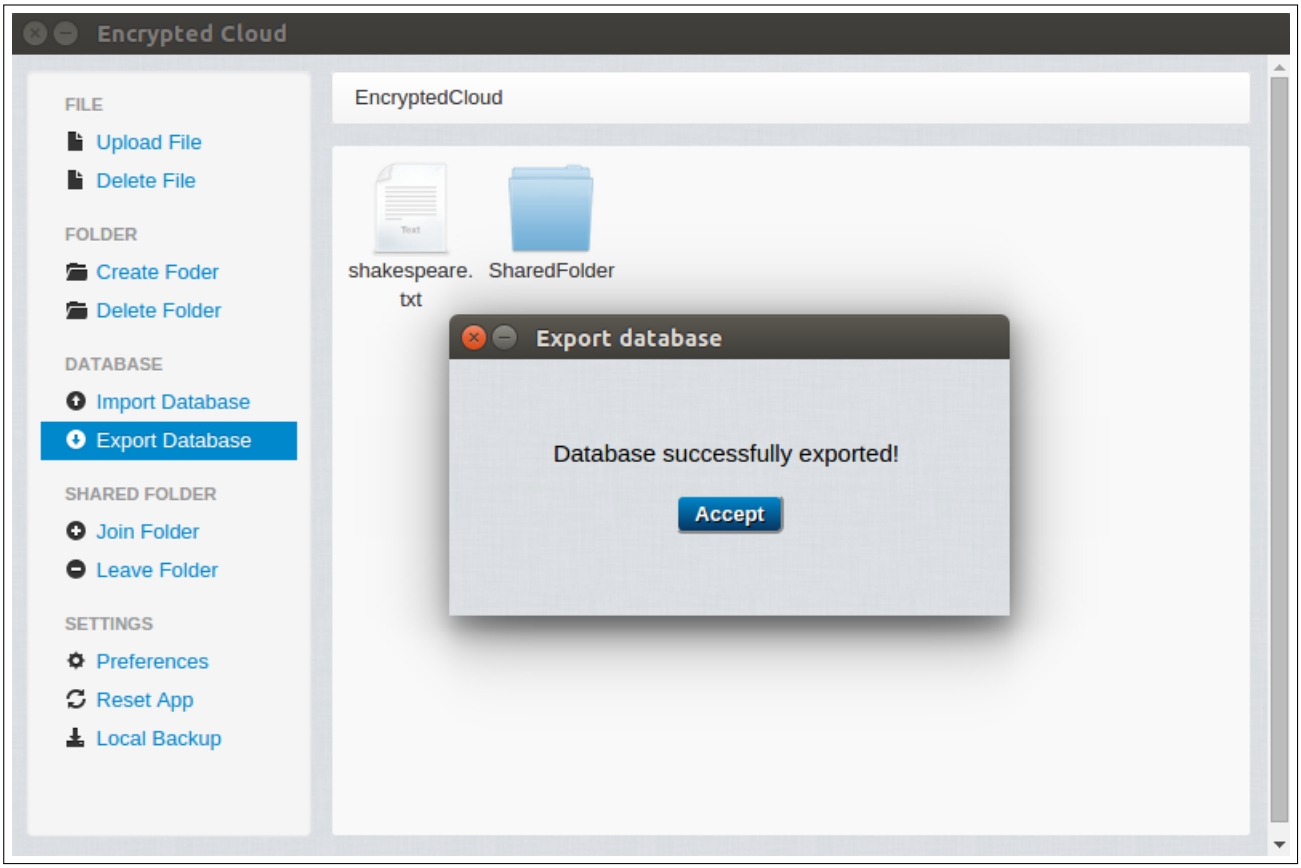


Figure 4.31: Export database pop-up.

4.6.2.3.11 Change User password

The user has the possibility to change her password. She has to click on the link "Preferences" on the left side panel which shows the following pop-up:

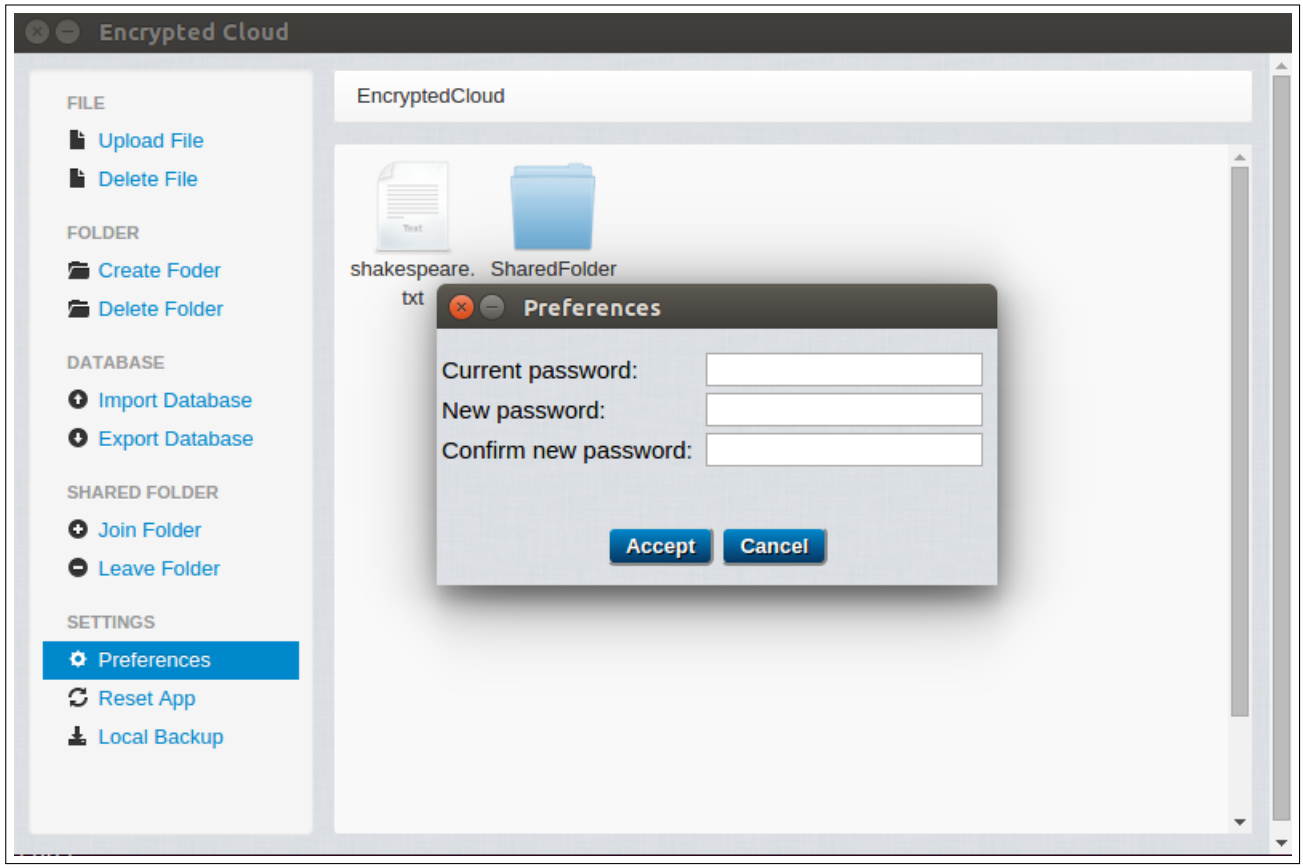


Figure 4.32: User password modification.

Then, the user has to provide the old password and type the new password twice. If these data are provided correctly, the password will be changed. In addition, it is also updated the database key and all non-shared files are re-encrypted because their keys depend on this user's password.

4.6.2.3.12 Encrypted Cloud reset

The user can reset Encrypted Cloud by clicking on the link "Reset App" on the left side panel. This action could be needed if the user wants to change the directories where she stores her assets or choose a different asset distribution protocol. This restart will delete both local database as well as all non-shared assets. However, before resetting Encrypted Cloud, the user has the option of making a local backup (see Section 4.6.2.3.13) of all her assets, both shared and non-shared. For this, she needs to select in a pop-up (see Figure 4.33) the directory where she wants to save her backup, and it has to be different than the directories selected in the set up of Encrypted Cloud. If the user does not select any directory, no backup will be made.

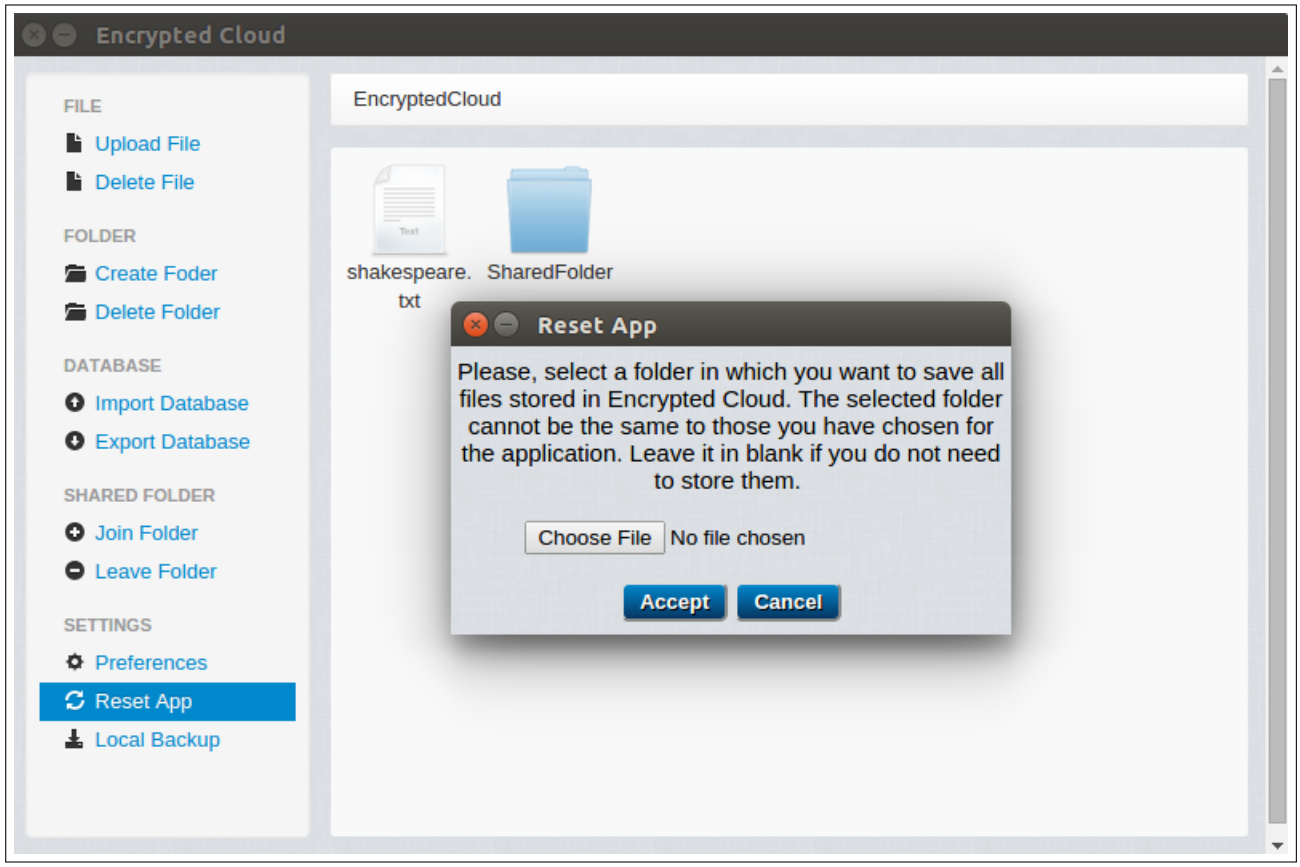


Figure 4.33: Encrypted Cloud reset.

4.6.2.3.13 Local backup

The user can make a local backup whenever she wants. She has to click on the link "Local Backup" on the left side panel. After that, the user has to select in a pop-up (see Figure 4.34) the directory where she wants to save her backup, and it has to be different from the directories selected in the set up of Encrypted Cloud. Then, in the chosen directory it is created a backup folder called *Encrypted_Cloud_Backup*. This backup functionality goes through all elements contained in both shared and non-shared directories, decrypting and saving each element in this backup folder. Note that the shared files are only decrypted if the user belongs to the shared folder.

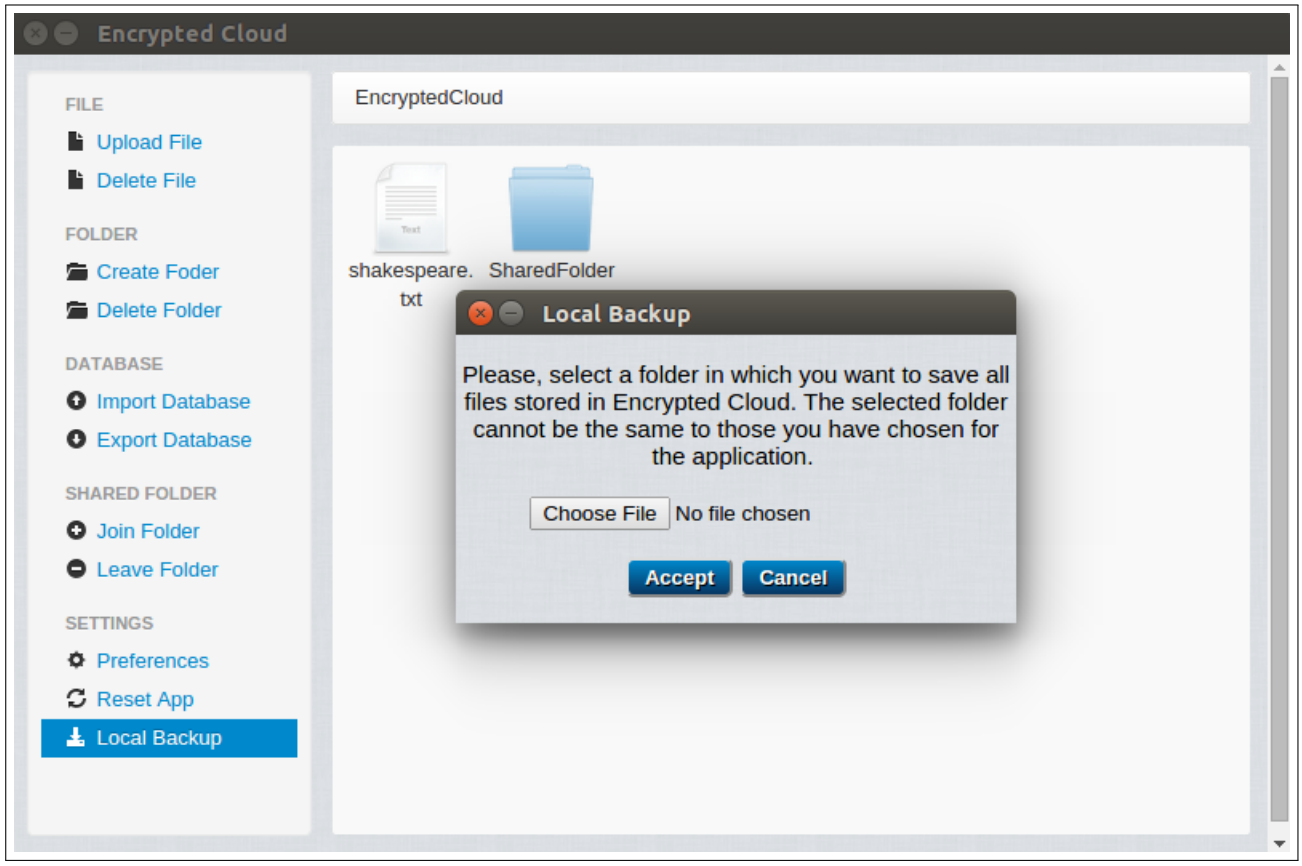


Figure 4.34: Encrypted Cloud local backup.

4.6.2.3.14 Monitoring of uploaded assets deleted or moved

The CP could decide to delete these files which are not accessed by the users for a long time in order to gain storage space. For this reason, Encrypted Cloud has a specific service for testing if any uploaded files has been deleted or moved by an unauthorised party.

To do this, every time that it is added a new asset to Encrypted Cloud, its path is stored in the Asset entity of the local database, as detailed in Section 4.6.2.3.4 and Section 4.6.2.3.7. This monitoring function applies to all non-shared files and to all shared files contained in these shared folders that are owned by the current user.

Therefore, each time the file explorer of Encrypted Cloud is loaded, an asynchronous task is started which verifies if all the assets stored in the Asset entity are in their actual paths. If not, the user will be notified, as it is depicted in Figure 4.35. Nonetheless, keep in mind that this behaviour is only notified, without the possibility of recovering the removed asset.

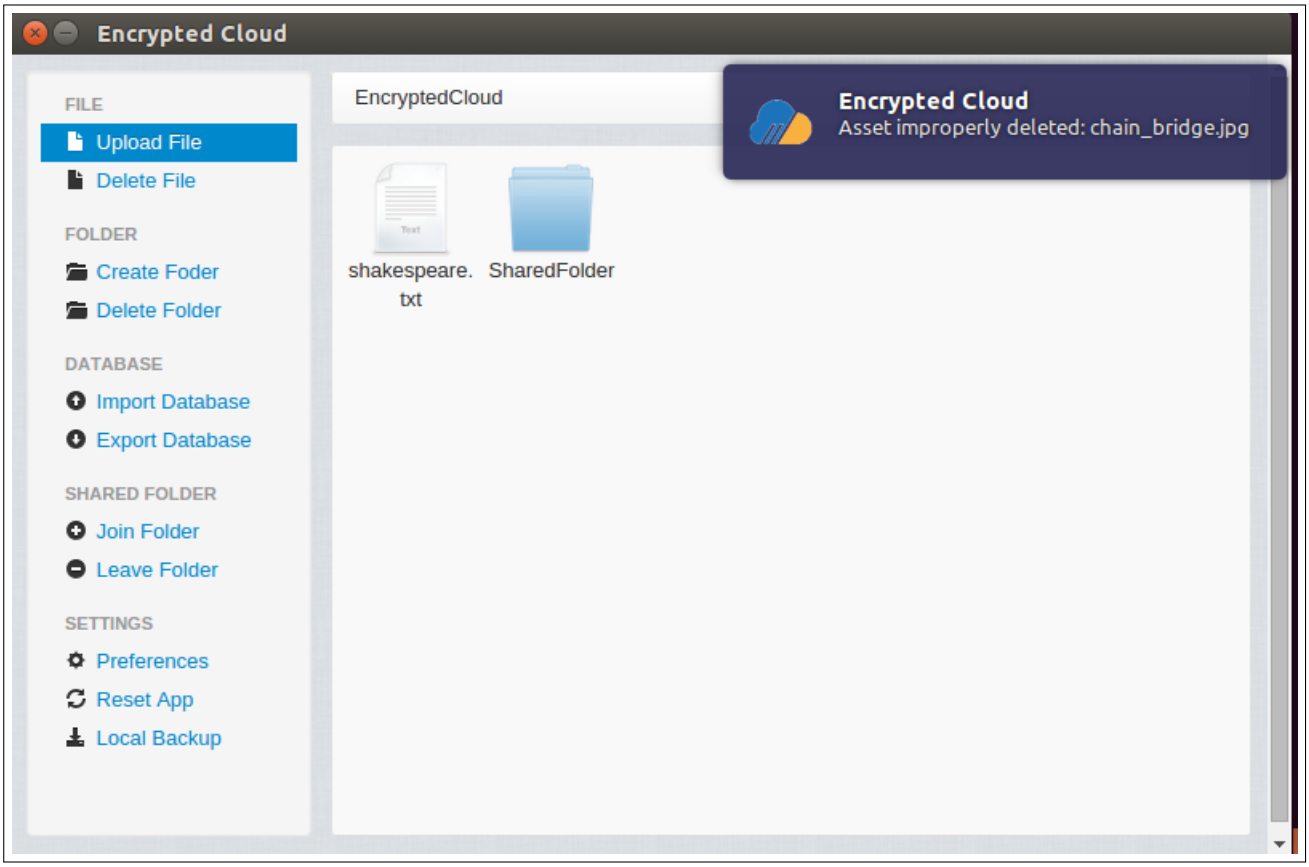


Figure 4.35: Notification of deleted file.

4.7 Test plan

Once the implementation of Encrypted Cloud was finished, the next step was to design a test plan and to carry it out, with the main objective of finding the maximum number of errors in order to correct them. The main rule followed is to only test these new functions created specifically for Encrypted Cloud, and not the third-party libraries used. These tests made can be classified in two different groups:

- **Unit tests:** check the smallest unit of functionality, typically a method/function and should be focussed on one particular feature.
- **Integration tests:** test the correct inter-operation of multiple subsystems by combining the units of code and testing that the resulting combination works correctly.

4.7.1 Unit tests

The unit tests made had the objective of checking the correct operation of all the individual features that do not depend on others for its proper functioning. Is was used *mocha* as a unit test tool along with the assertion library *chai*.

4.7.1.1 Unit tests carried out

This subsection details each unit test carried out group by its library. Note that some libraries can only be tested through integration tests.

- **crypto_library.js**
  - **createRSA512KeyPairNotNull:** checks that a 512 bits RSA key is created and it is not null.
  - **publicRSAKeyToString:** checks that a 512 bits RSA public key is converted to a string.
  - **privateRSAKeyToString:** checks that a 512 bits RSA private key is converted to a string.
  - **setGlobalRSAPublicKey:** checks that a global variable is set to the value of a 512 bits RSA public key.
  - **setGlobalRSAPrivateKey:** checks that a global variable is set to the value of a 512 bits RSA private key.
  - **createSeedAESKey256NotNull:** checks that a 256 bits seeded AES key is created and it is not null.
  - **createIV16NotNull:** checks that a 16 bytes initialisation vector is created and it is not null.
  - **encryptDecryptAES256:** checks that a plain text is encrypted and decrypted with AES 256 CBC.
  - **createHashNotNull:** checks that a hash is created and it is not null.
- **database_library.js**
  - **checkDBConnectionOK:** checks that a database connection is successful.
  - **checkDBConnectionERROR:** checks that a database connection is erroneous due to an incorrect database password.

- **checkUpdateDBPassword:** checks that the database password is updated with a new password.
- **checkInsertEncryptedCloud:** checks that a register is inserted into EncryptedCloud entity.
- **checkInsertAsset:** checks that a register is inserted into Asset entity.
- **checkInsertRequest:** checks that a register is inserted into SharedFolderRequest entity.
- **checkInsertSharedFolderUsers:** checks that a register is inserted into SharedFolderUsers entity.
- **checkDeleteAsset:** checks that a register is deleted from Asset entity.
- **checkDeleteRequest:** checks that a register is deleted from SharedFolderRequest entity.
- **checkGetFilesUploadedFromDB:** checks that a register is read from Asset entity.
- **checkGetNonSharedAssetsFromDB:** checks that a non-shared asset is read from Asset entity.
- **checkReadRequestsFromDB:** checks that a request is read from SharedFolderRequest entity.
- **checkUpdateUsersSharedFolder:** checks that a register is updated in SharedFolderUsers entity.
- **checkGetUsersSharedFolder:** checks that a register is read from SharedFolderUsers entity.
- **checkDeleteUsersSharedFolder:** checks that a register is deleted from SharedFolderUsers entity.
- **checkUpdateAssetProtocol:** checks that a register is updated into EncryptedCloud entity.

- **fs_library.js**
  - **rmDirContents:** checks that the contents of a directory are removed.
  - **rmDirItself:** checks that it is removed all inside a directory and the directory itself.
  - **getPathLessSize:** checks that the path with less size occupied it is obtained from an array of paths.

- **getDirsRelativePath:** checks that the relative path of a file from an array of directories is obtained. For example, the relative path of "/home/Dropbox/alex.txt" respect to two folders "/home/Dropbox" and "/home/MEGA" is "alex.txt".
- **getDirsRootPath:** checks that the root path of a file from an array of directories is obtained. For example, the root path of "/home/Dropbox/alex.txt" respect to two folders "/home/Dropbox" and "/home/MEGA" is "/home/Dropbox".
- **getFiles:** checks that the contents of a directory are read except hidden elements.

- **shared_folder_library.js**
  - **createUsersFile:** checks that *.usersFolder.txt* file is created.
  - **createUpdatingFile:** checks that *.updatingFolder.txt* file is created.

4.7.1.2 Unit tests results

At the end, a set of 33 unit tests was performed. At the beginning, several errors were obtained so they had to be fixed. Finally, all unit tests were passed as is shown in Appendix B.

4.7.2 Integration tests

Integration tests check all the functionality provided by the application which have not been previously verified by unit tests. Therefore, the main goal of this kind of tests is to assure that all libraries presented before are integrated without problems. These tests had to be done manually, and several virtual machines were needed to complete the whole set of tests. All tests were done in both Ubuntu and Windows, being very useful the *node-notifier*[17] package of Node.js, which is a module for sending cross-platform system notification.

The following subsections present the four main integration tests. However, the Appendix C contains other integration tests made.

4.7.2.1 Upload file integration test

In this section there are different tests related to uploading a file to Encrypted Cloud. To perform this action, first, the user has to click on the link "Upload File". Then,

[17] www.npmjs.com/package/node-notifier

the user has to select the file she wants to upload through a file explorer. Finally, she should click on the "Open" button to complete this action.

The first test consists of uploading two files to the root path of the application (non-shared folder). To do this, the user clicks on "Upload File" and then selects the two files she wants to upload. Finally, she presses the "Open" button:

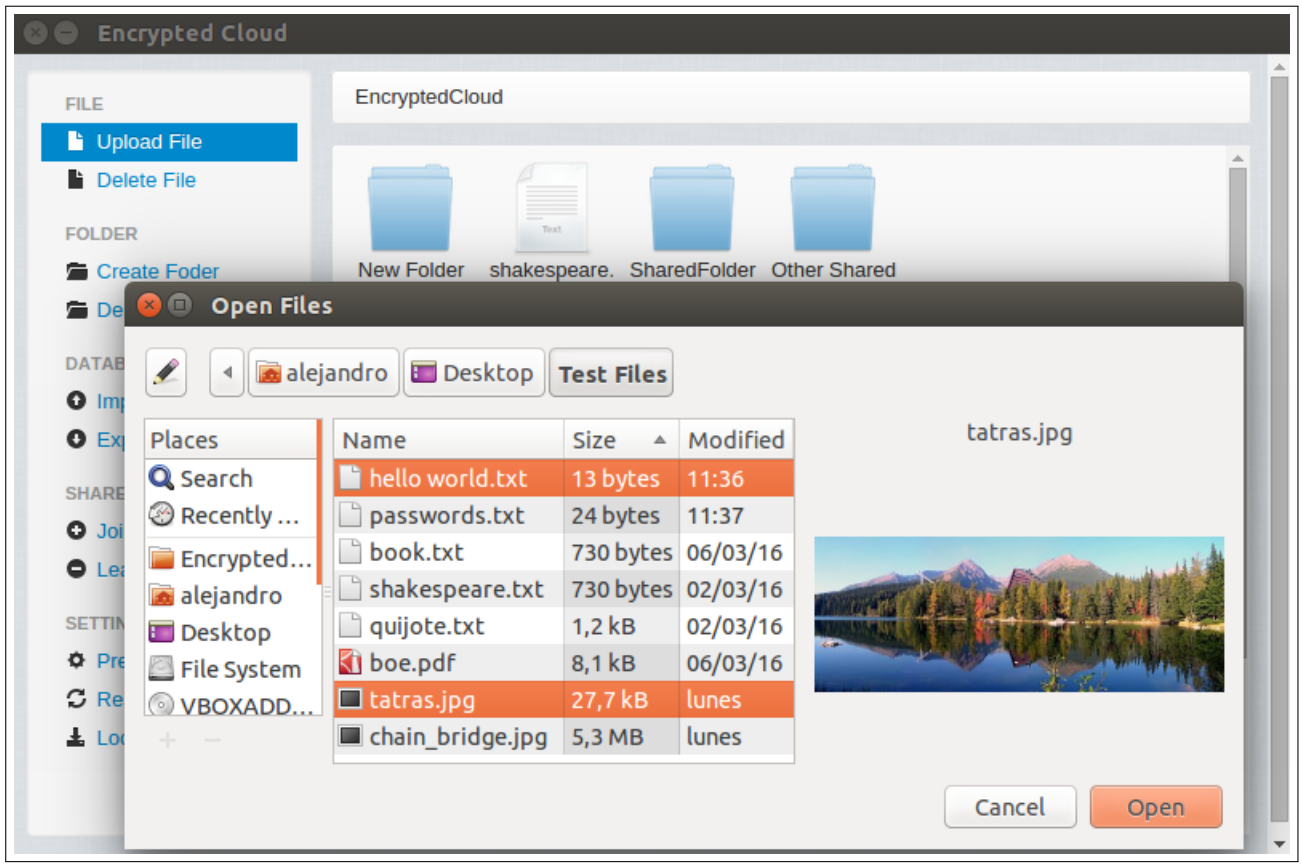


Figure 4.36: Upload file integration test (I).

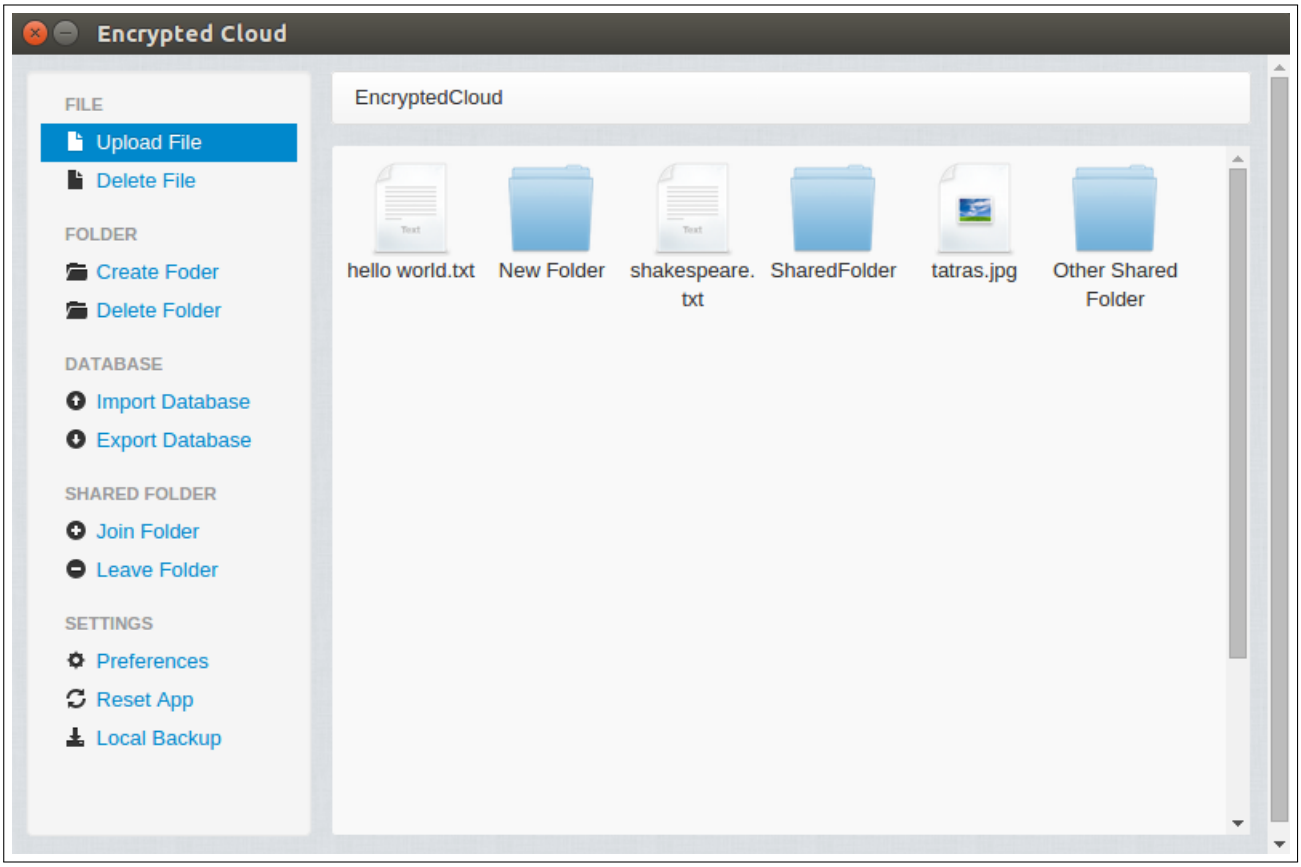


Figure 4.37: Upload file integration test (II).

In the second test, the user uploads a file to a shared folder to which she belongs. She has to be previously joined to this shared folder and to stay in the folder:

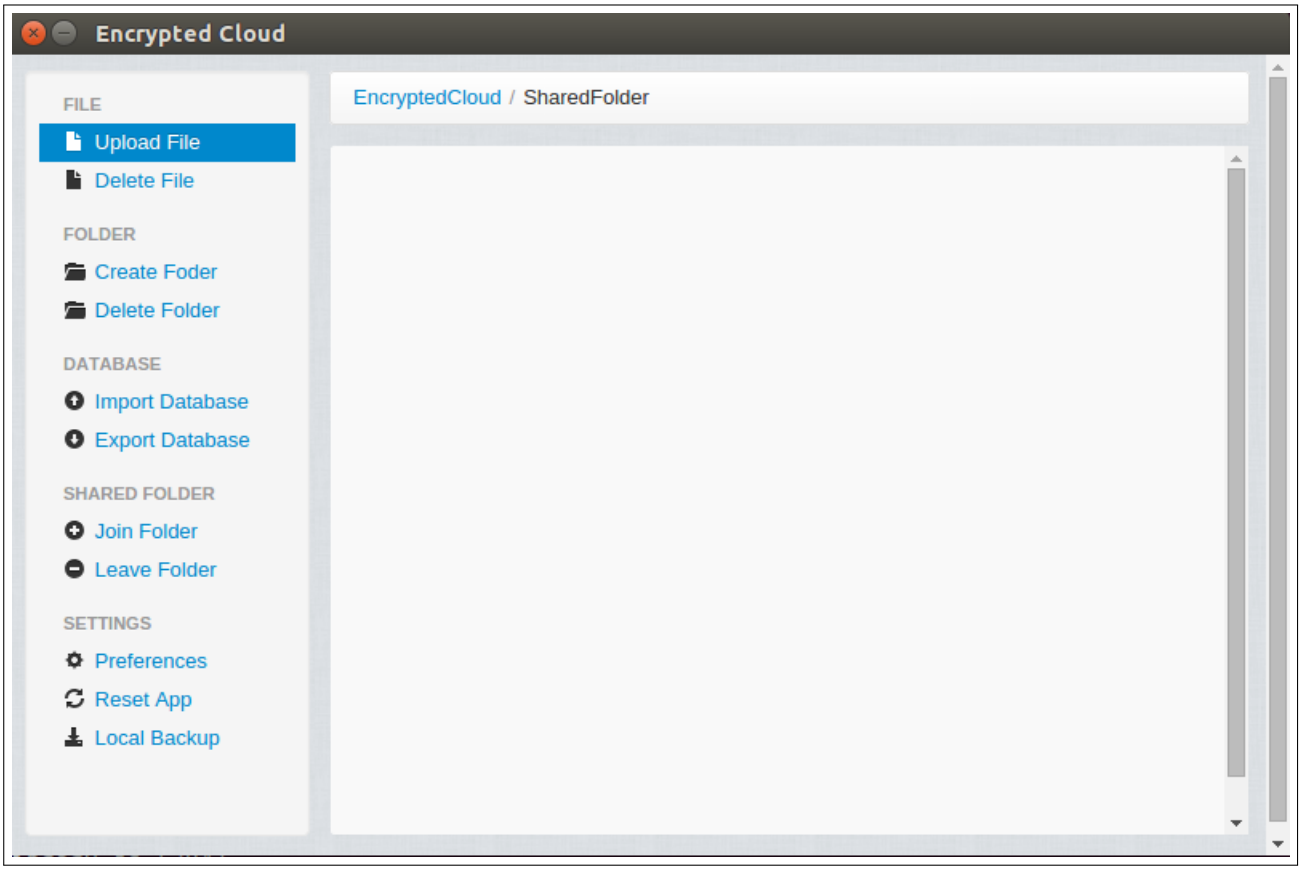


Figure 4.38: Upload file integration test (III).

Then, the user selects the file she wants to upload and presses on the "Open" button, verifying that the file has successfully uploaded, as seen in Figure 4.39 and in Figure 4.40.

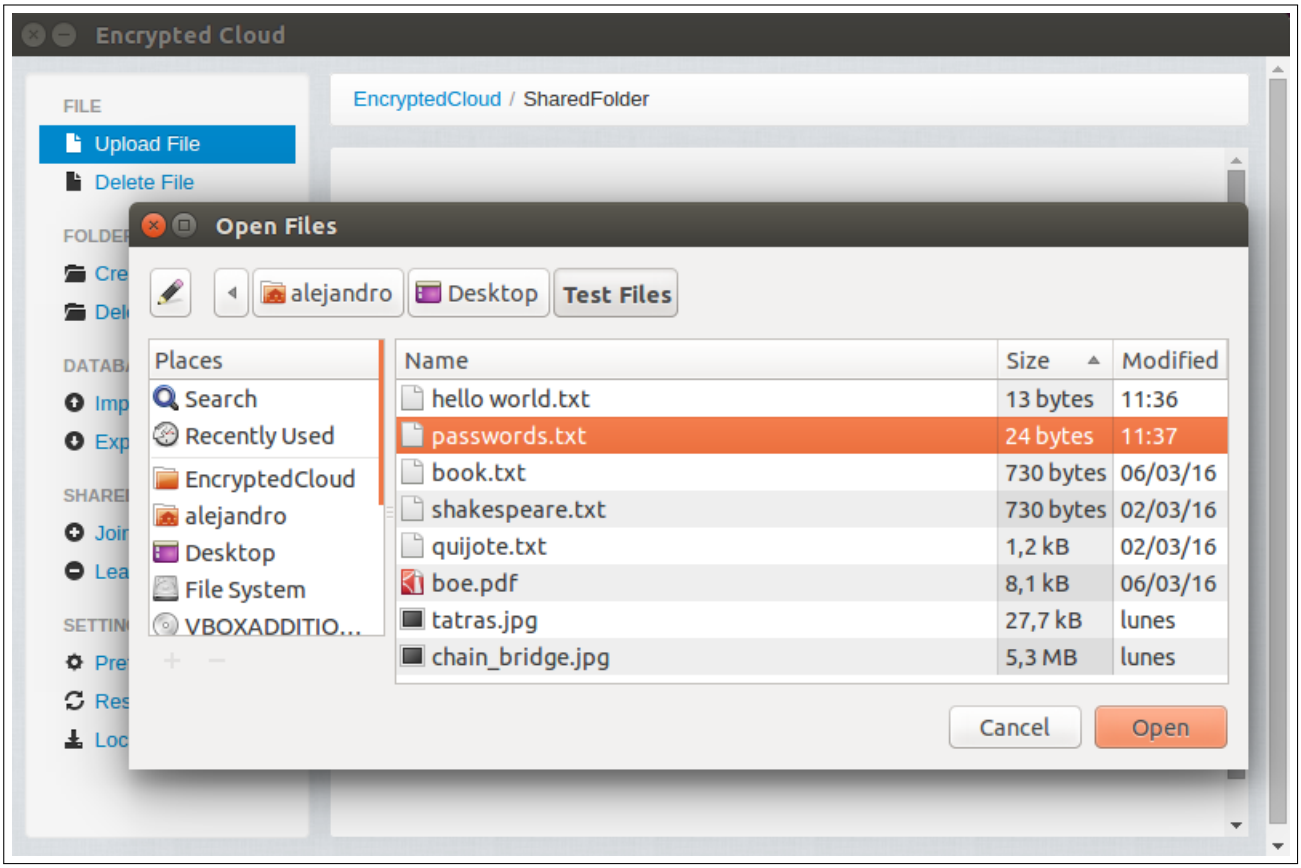


Figure 4.39: Upload file integration test (IV).

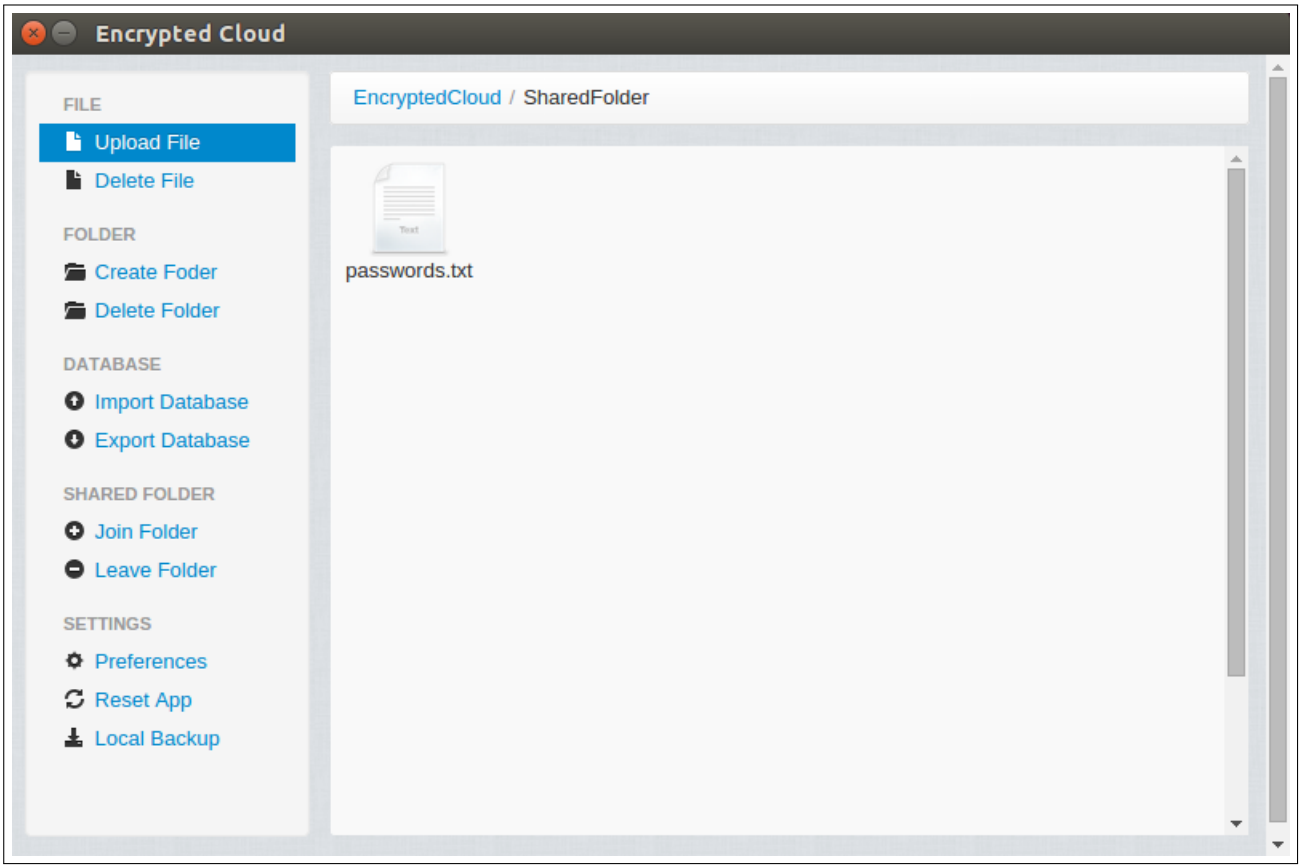


Figure 4.40: Upload file integration test (V).

The next test verifies that a user cannot upload a file to a shared folder to which she does not belong. In this case, the user tries to upload a file in the shared folder "Other Shared Folder", and as she is not joined to this folder, a pop-up notifies the action (Figure 4.41).

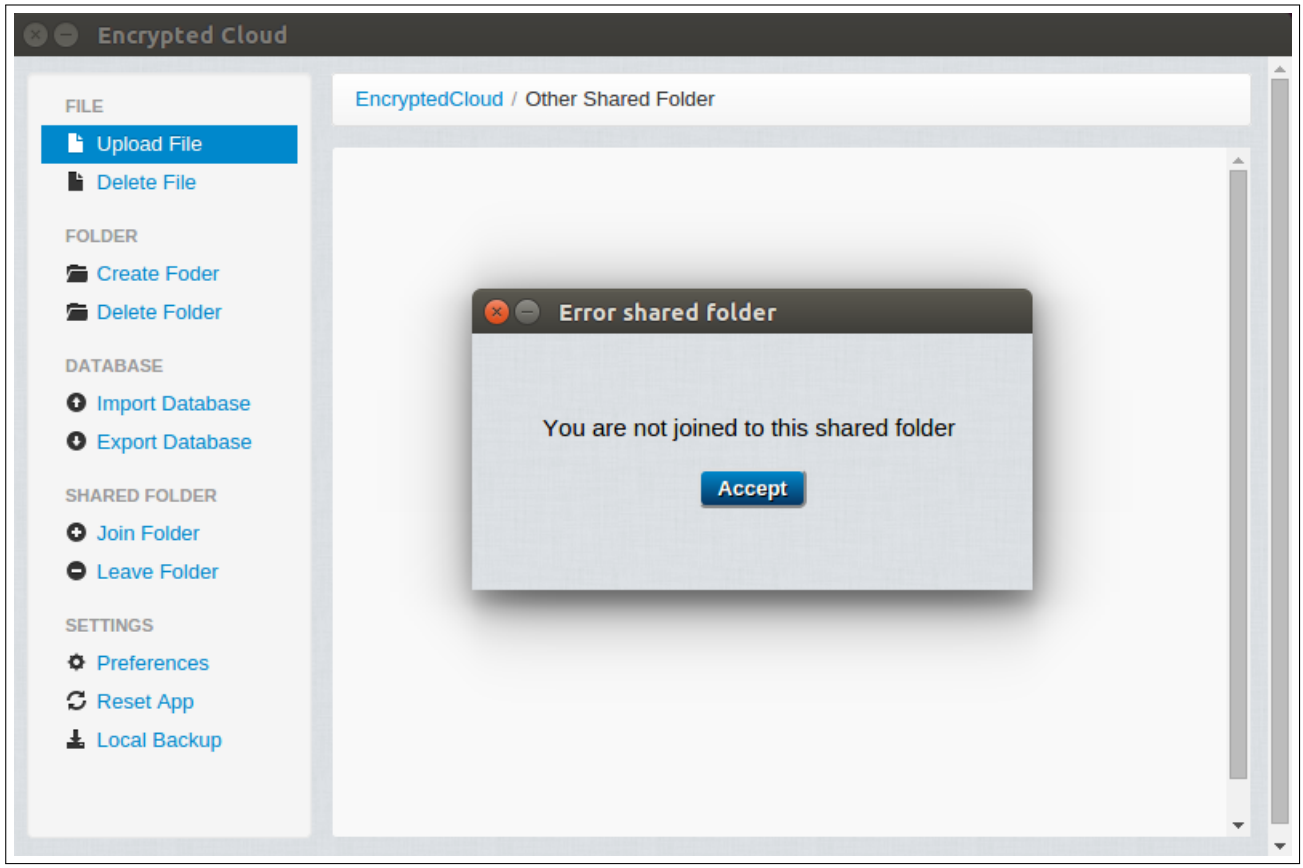


Figure 4.41: Upload file integration test (VI).

Finally, remind that if a shared folder is being updated by its owner, it is not allowed to upload a file to that folder until the update finishes. In this test, the user tries to perform the same steps as in the second test, but in this case the folder is being updated (this behaviour is indicated by the presence of a file called *.updatingFolder.txt* in the shared folder, as shown in Figure 4.42), so the user cannot upload her file:

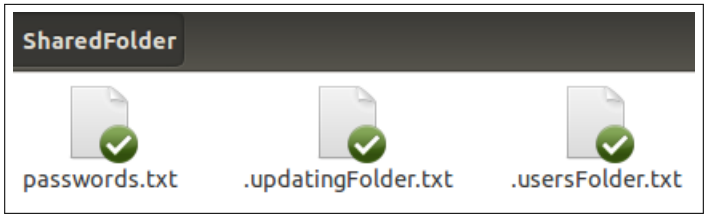


Figure 4.42: Upload file integration test (VII).

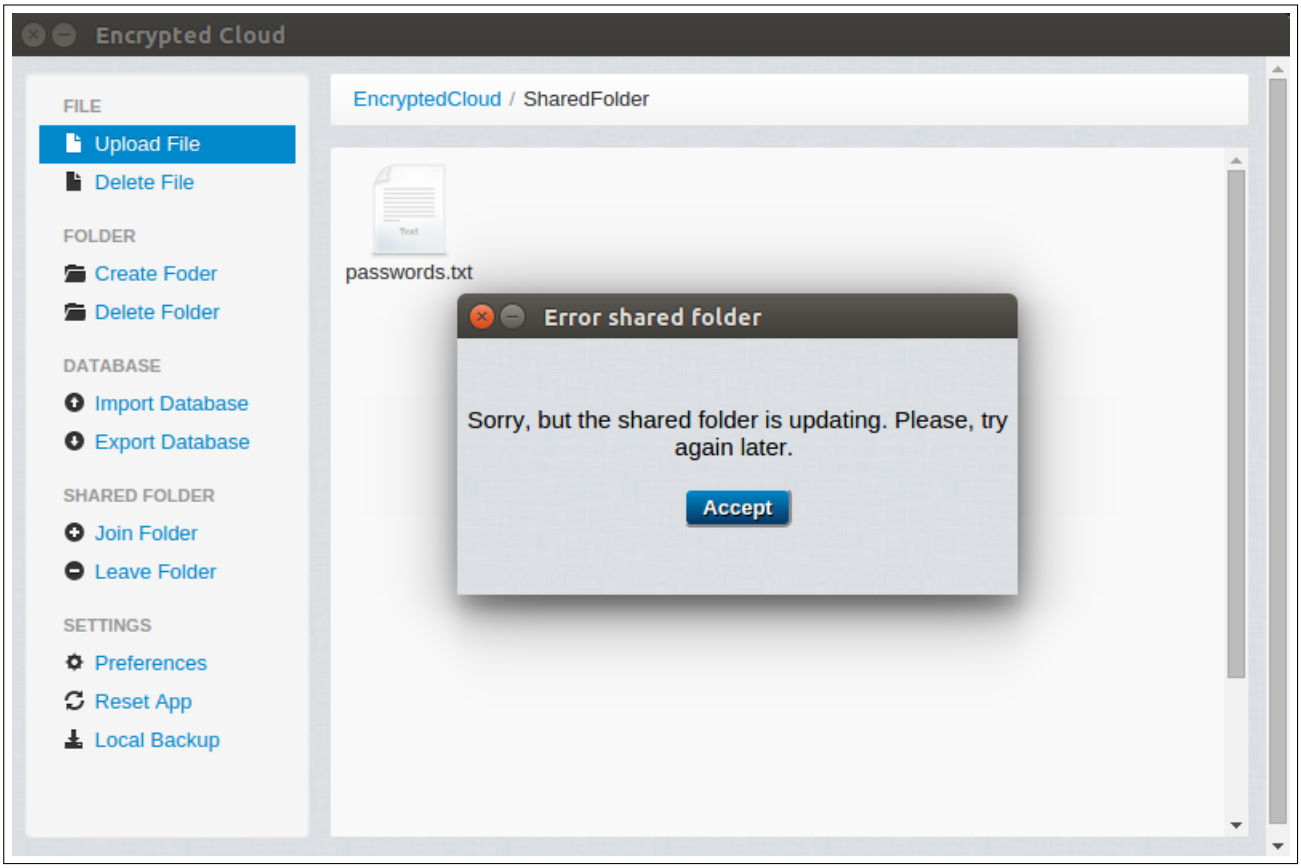


Figure 4.43: Upload file integration test (VIII).

4.7.2.2 View file integration test

This section shows different tests for the functionality of viewing a file from Encrypted Cloud. Remember that to open a file, the user has only to perform a double click on it. In the first test, the user opens a file previously uploaded to a non-shared folder, called "tatras.jpg" (see Figure 4.44).

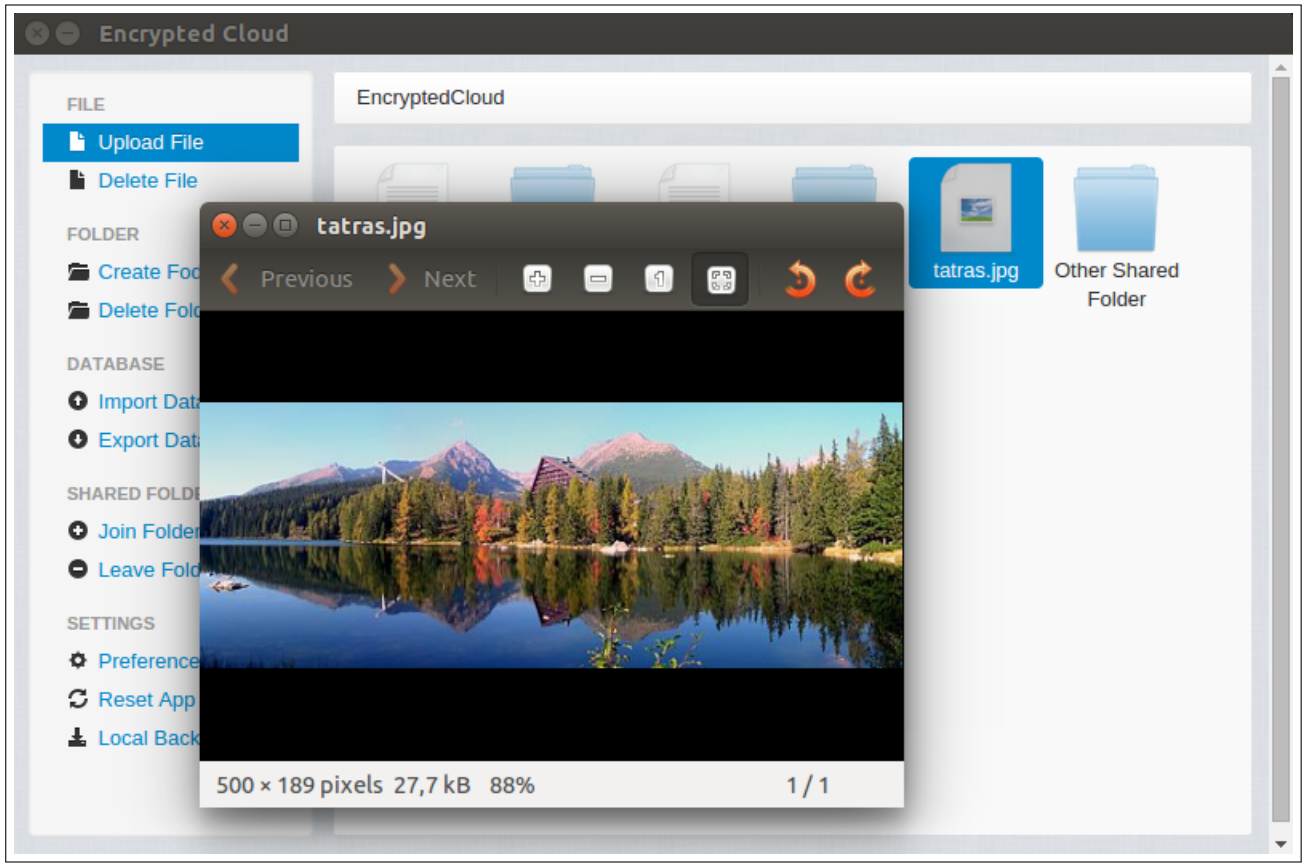


Figure 4.44: View file integration test (I).

The second test displays a file stored in a shared folder to which the user is joined. First, the user enters in "Shared Folder" and performs a double click on the "password.txt" file, showing its contents into the text editor:

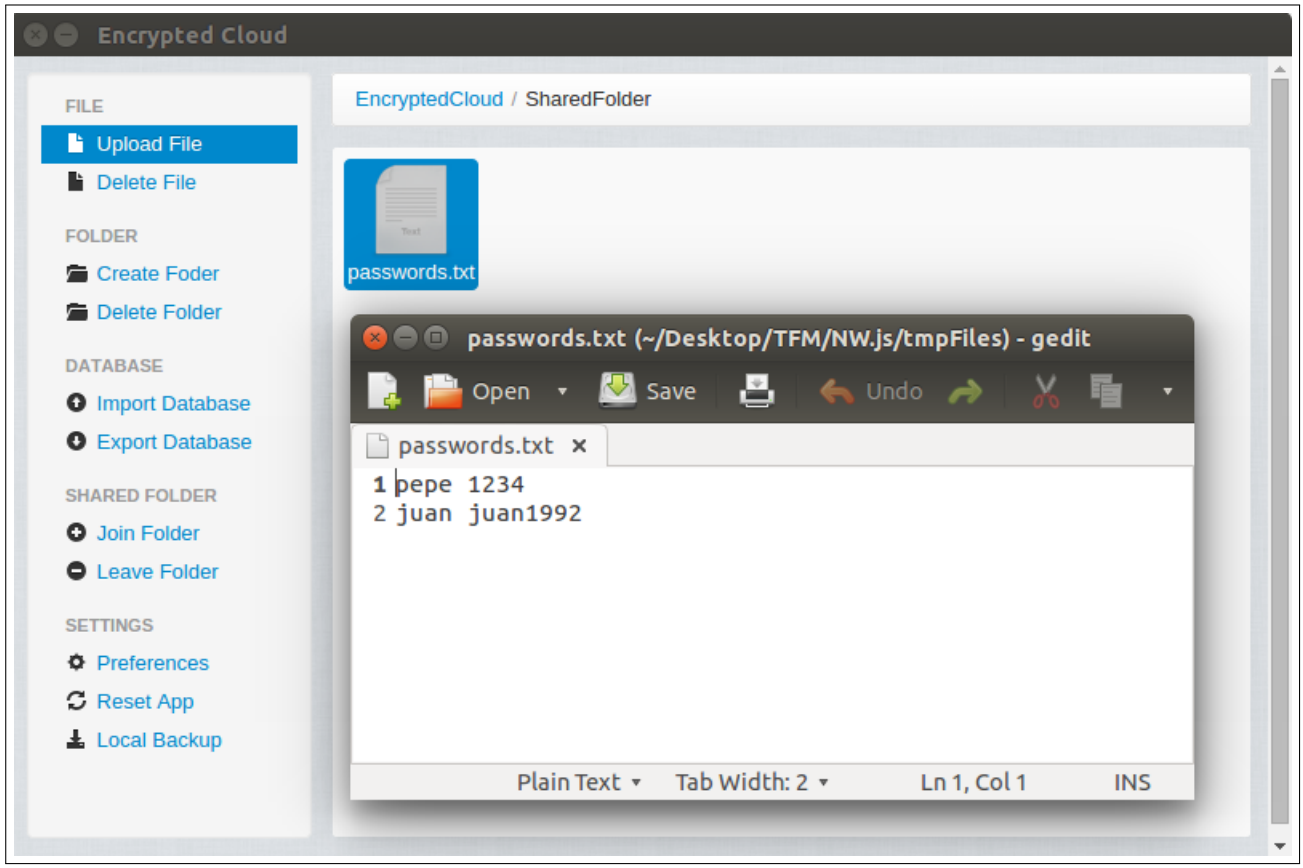


Figure 4.45: View file integration test (II).

The next test checks that a user cannot view a file in a shared folder to which she does not belong. The user navigates to "Other Shared Folder" and tries to view the "book.txt" file without success:

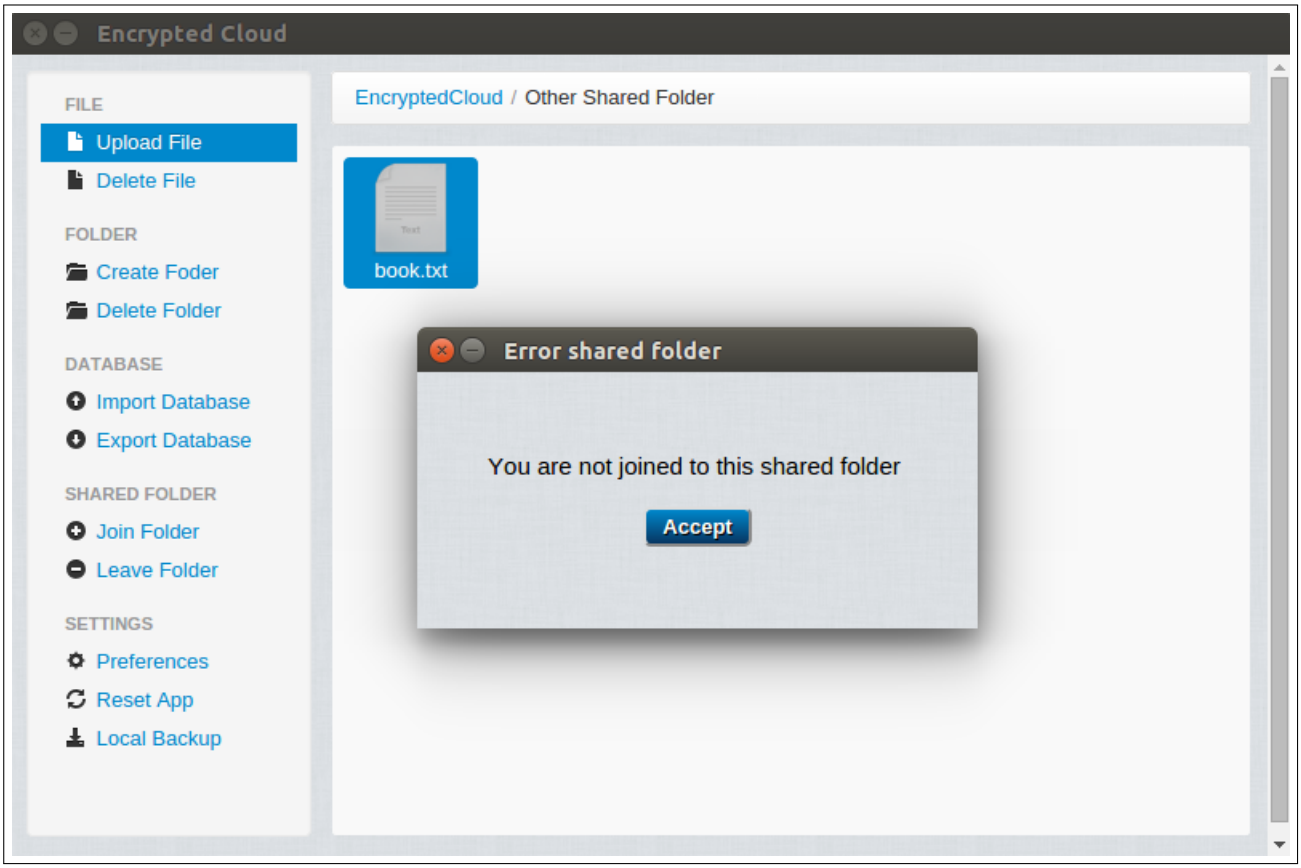


Figure 4.46: View file integration test (III).

The fourth test verifies that when a shared folder is being updated by its owner, the user cannot view a file stored in that folder until the update finishes. In this test, the user attempts to perform the same steps as in the second test, but in this case she is not able to view the file:

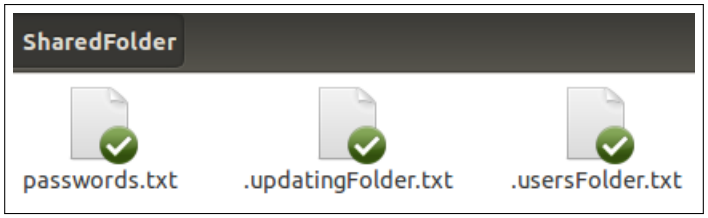


Figure 4.47: View file integration test (IV).

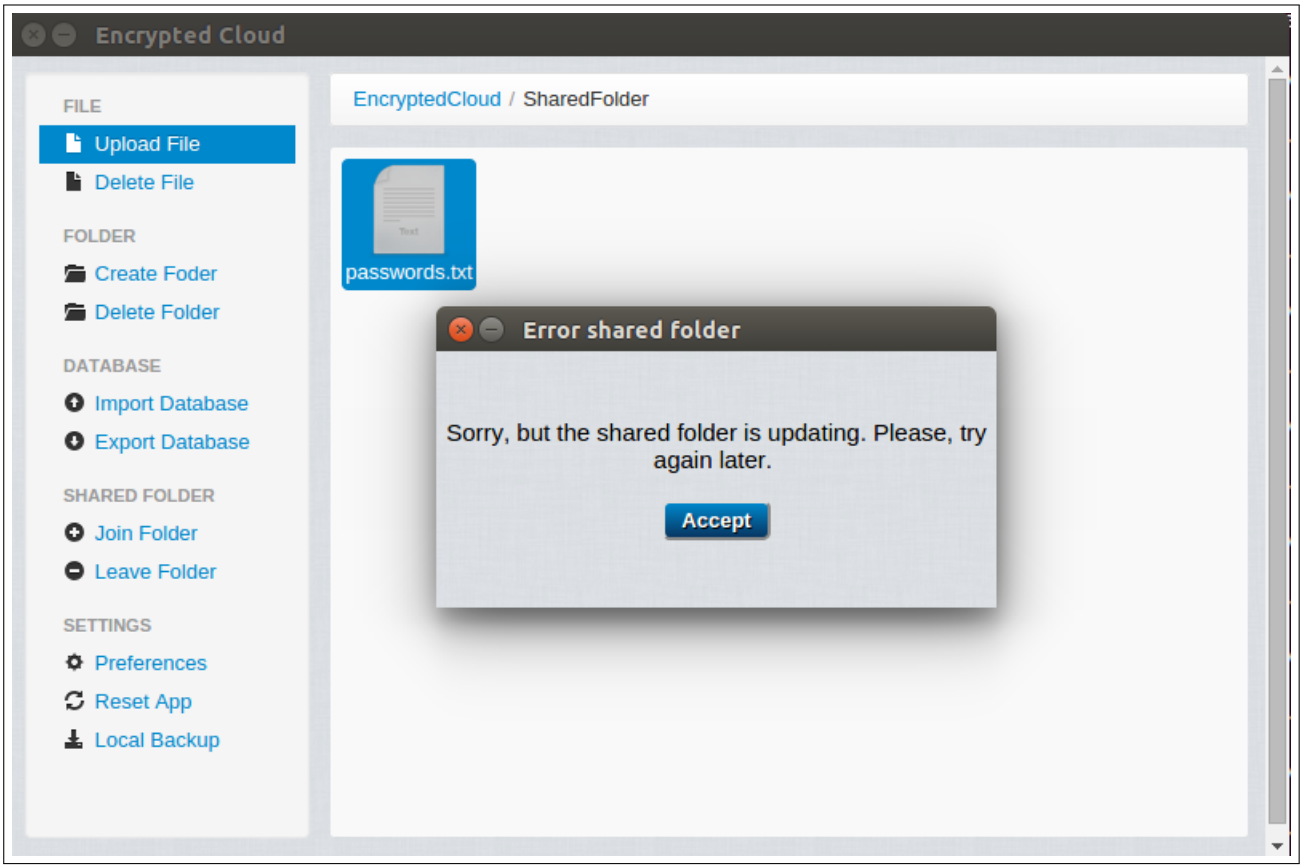


Figure 4.48: View file integration test (V).

Finally, Encrypted Cloud assures that when there is an error in the digital signature of a file, the user is notified when she tries to open it. To test this specific case, first an external file, "download.jpg", is manually added in one of the non-shared directories that the user selected in the set up of Encrypted Cloud. Then, the user tries to view the file, but a pop-up (Figure 4.49) informs her that it is corrupted.

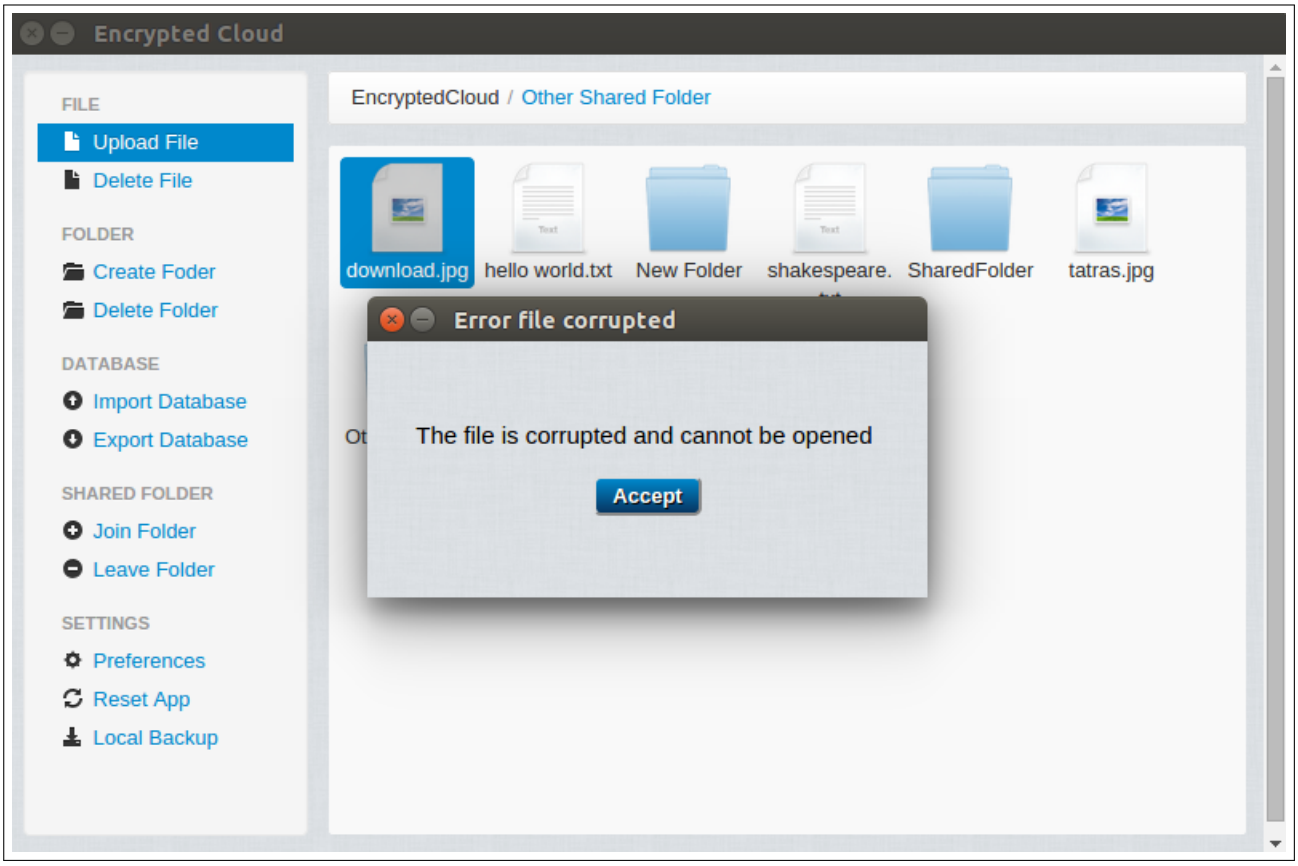


Figure 4.49: View file integration test (VI).

The last test verifies that all files displayed are temporarily stored in the *tmpFiles* folder:

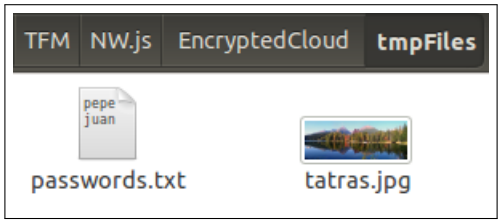


Figure 4.50: View file integration test (VII).

4.7.2.3 Join shared folder integration test

This section presents different tests of the functionality of joining to a shared folder. To join a shared folder, the user must first select the shared folder and then click on the link "Join Folder". The first test verifies that when a shared folder is being updated, the user cannot join it. In this example, the user fails trying to join the

"Other Shared Folder" that was being updated at this time, as shown in Figure 4.51 and in Figure 4.52.

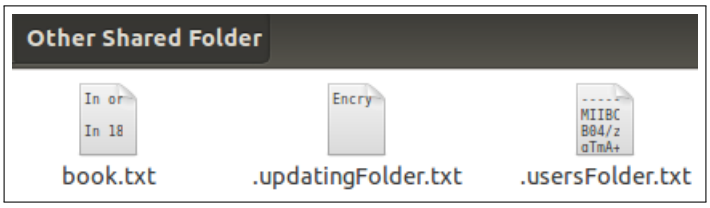


Figure 4.51: Join folder integration test (I).

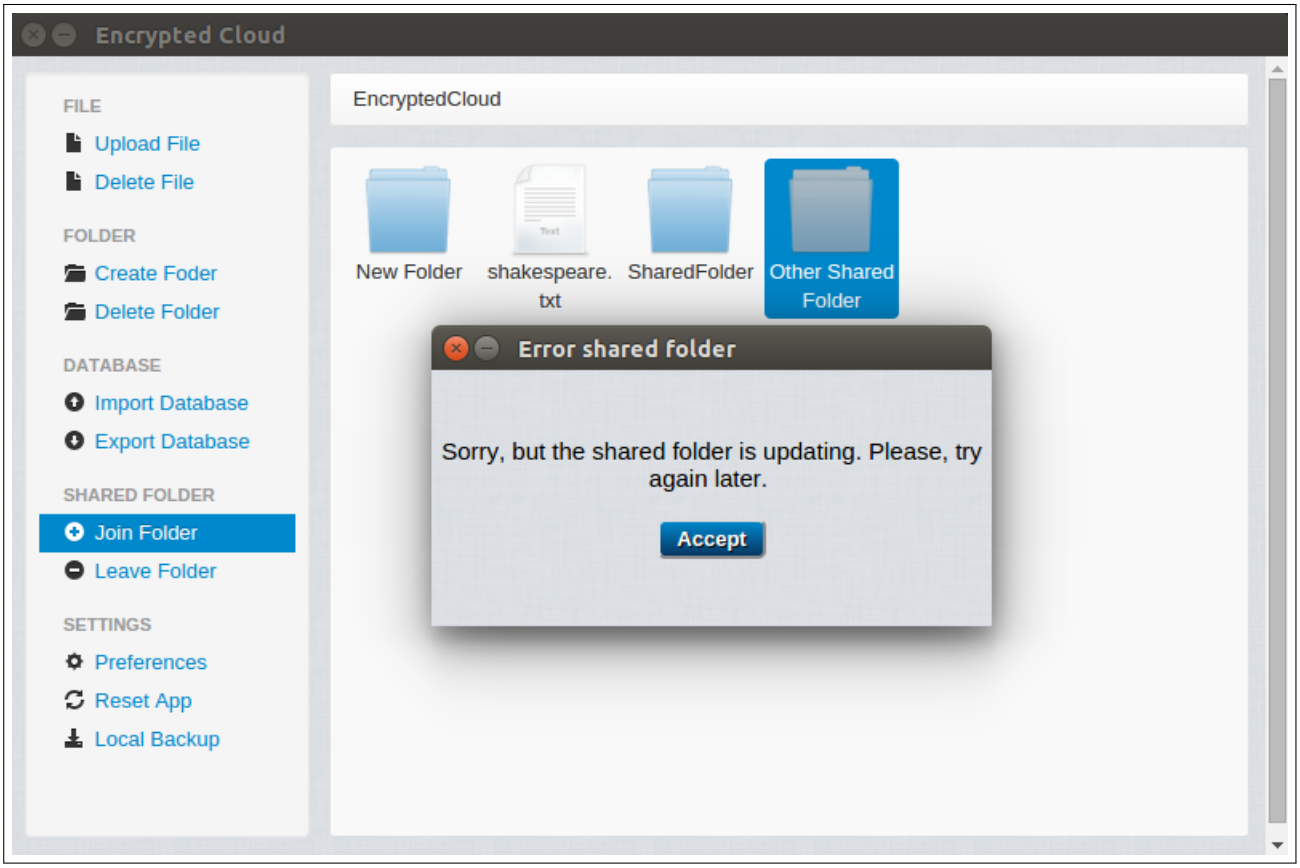


Figure 4.52: Join folder integration test (II).

When the updating of this shared folder finishes, the following tests are carried out, in which there are two different roles: the first is when the user that is going to join the shared folder will be owner, whereas the second is when she is going to make a join request in order to join the shared folder.

First, a user is joined to a shared folder as owner. For this, she selects the "Other Shared Folder", and she is notified that she has joined to the folder as owner (see

Figure 4.53).

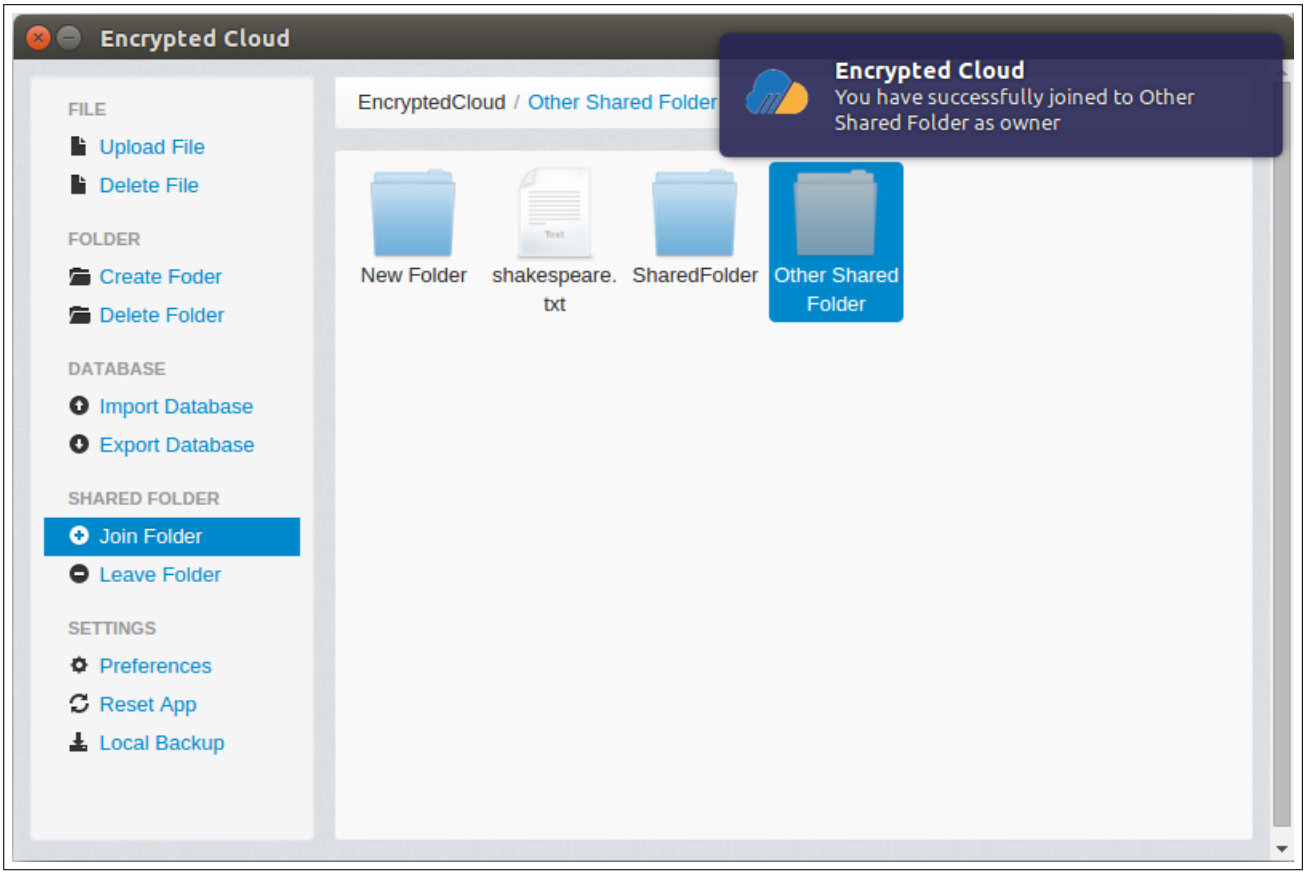


Figure 4.53: Join folder integration test (III).

After that, the procedure performed in the previous test is repeated in order to verify that a user cannot join twice to the same shared folder:

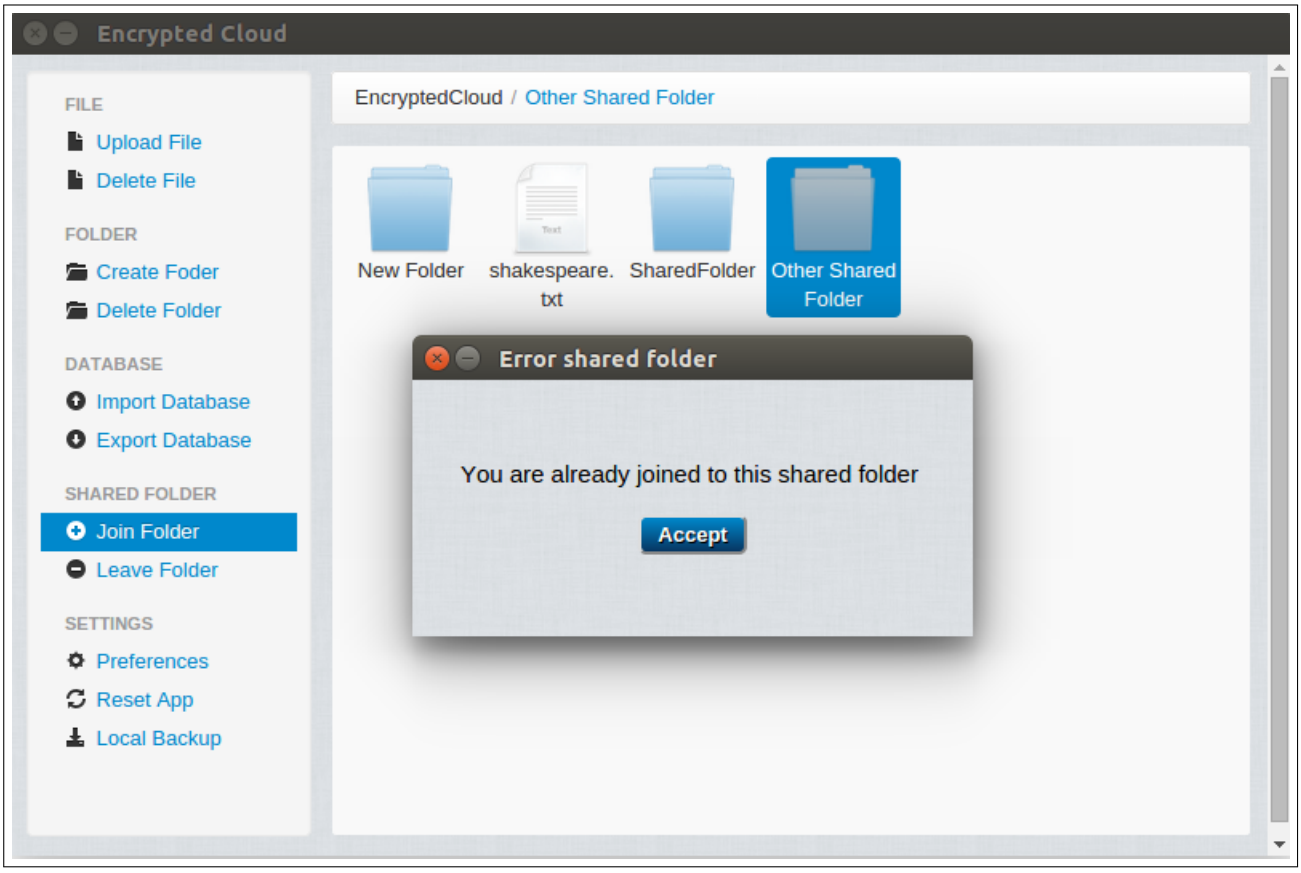


Figure 4.54: Join folder integration test (IV).

On the other hand, from the point of view of a user who will not be the owner of the shared folder, this test checks that the user sends a request to the owner, uploading it to a folder called *.join_requests* contained in the shared folder that she wants to join. In this test, the "Shared Folder" is selected in order to join it, as shown in Figure 4.55 and in Figure 4.56.

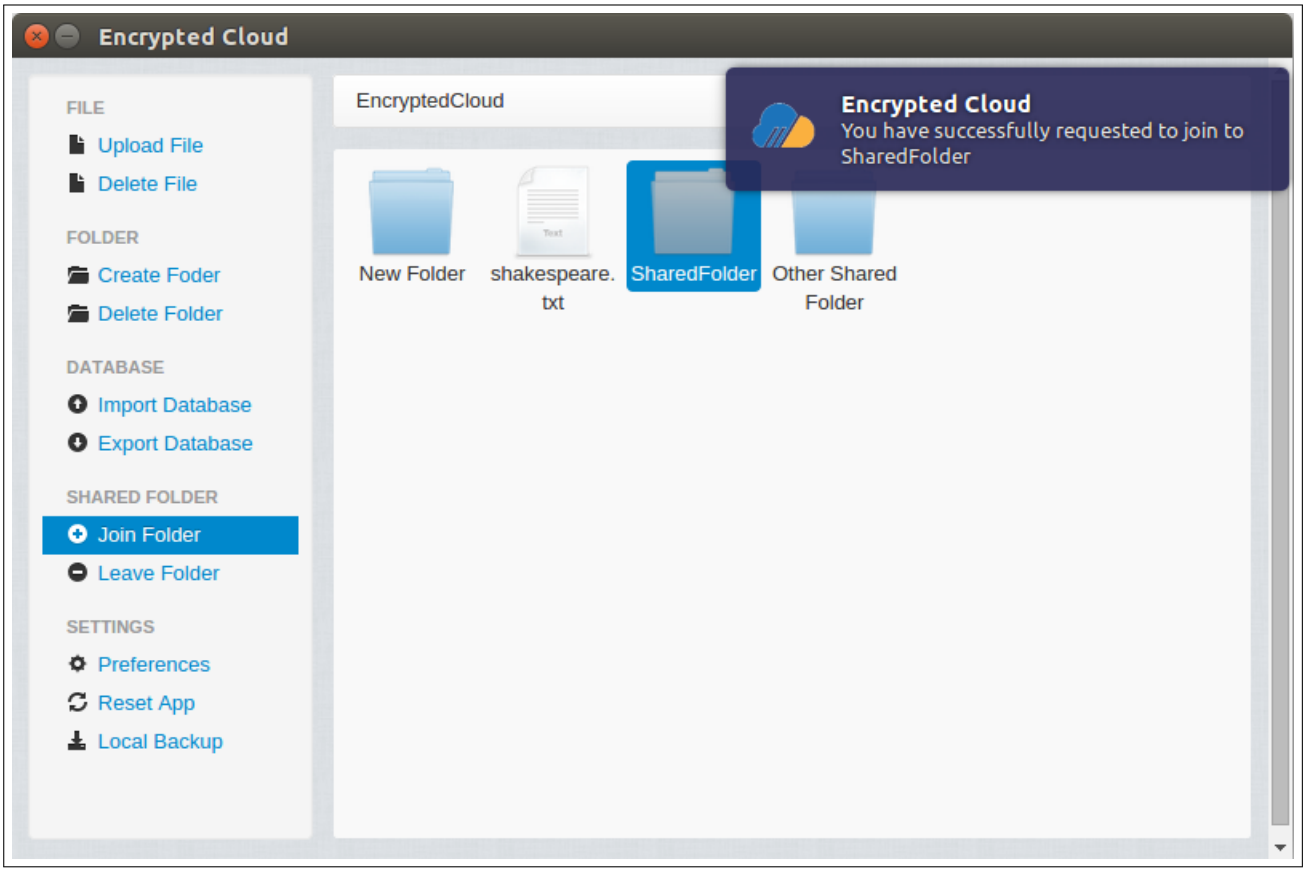


Figure 4.55: Join folder integration test (V).

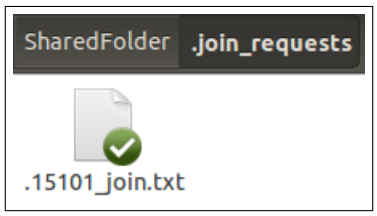


Figure 4.56: Join folder integration test (VI).

Besides, it is checked that if the request is deleted without being managed, it is uploaded again. This confirms that the requests are stored in the database in order to monitor them:

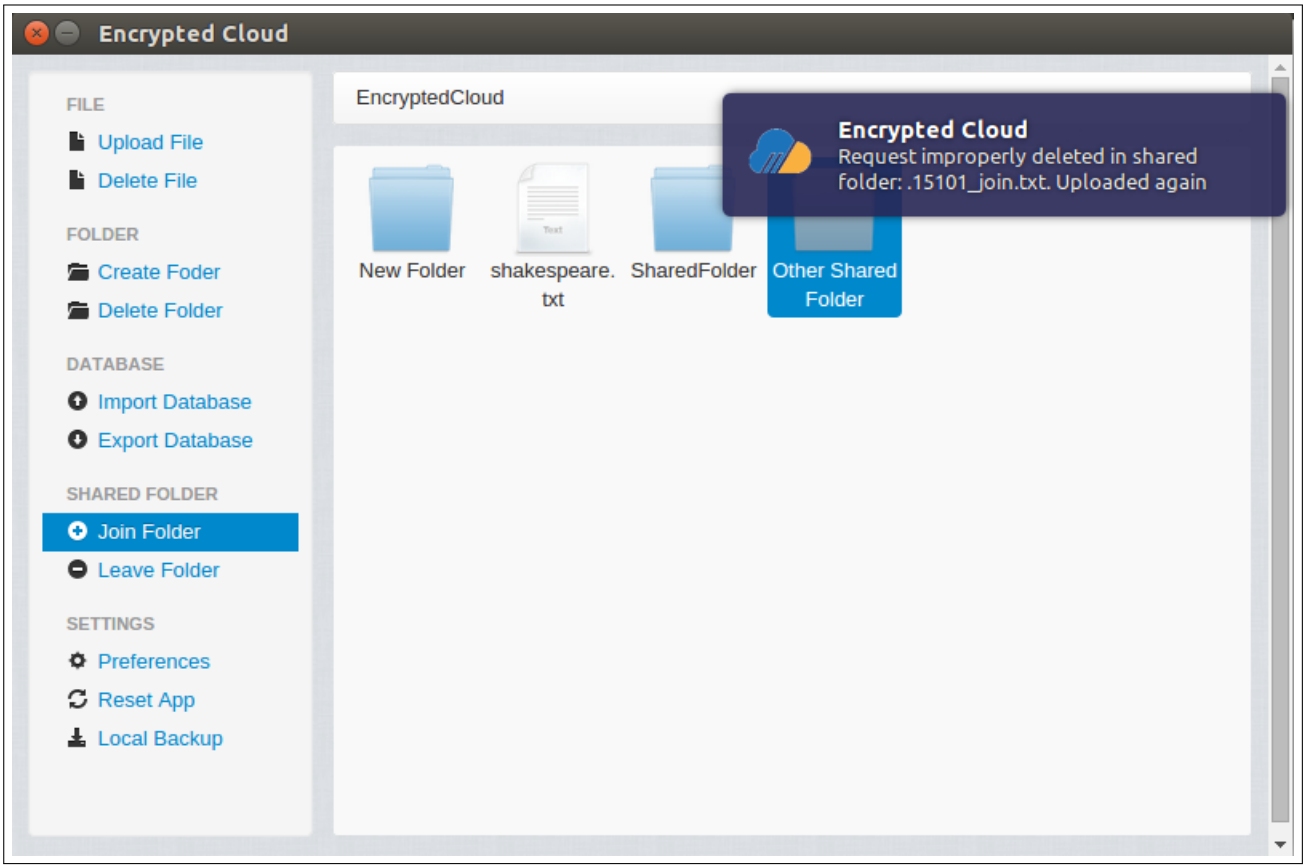


Figure 4.57: Join folder integration test (VII).

4.7.2.4 Leave shared folder integration test

This subsection tests the functionality of leaving a shared folder. This is a simple operation, since the user only has to select the folder she wants to leave and click on "Leave Folder".

To begin with, it is tested that when a shared folder is being updated, she cannot leave it. To carry out this test, the "Other Shared Folder", which is being updated at this moment, is selected. This process is illustrated in Figure 4.58 and in Figure 4.59.

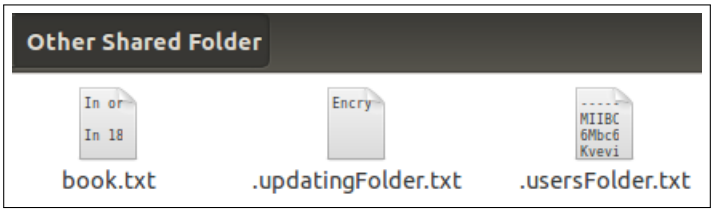


Figure 4.58: Leave folder integration test (I).

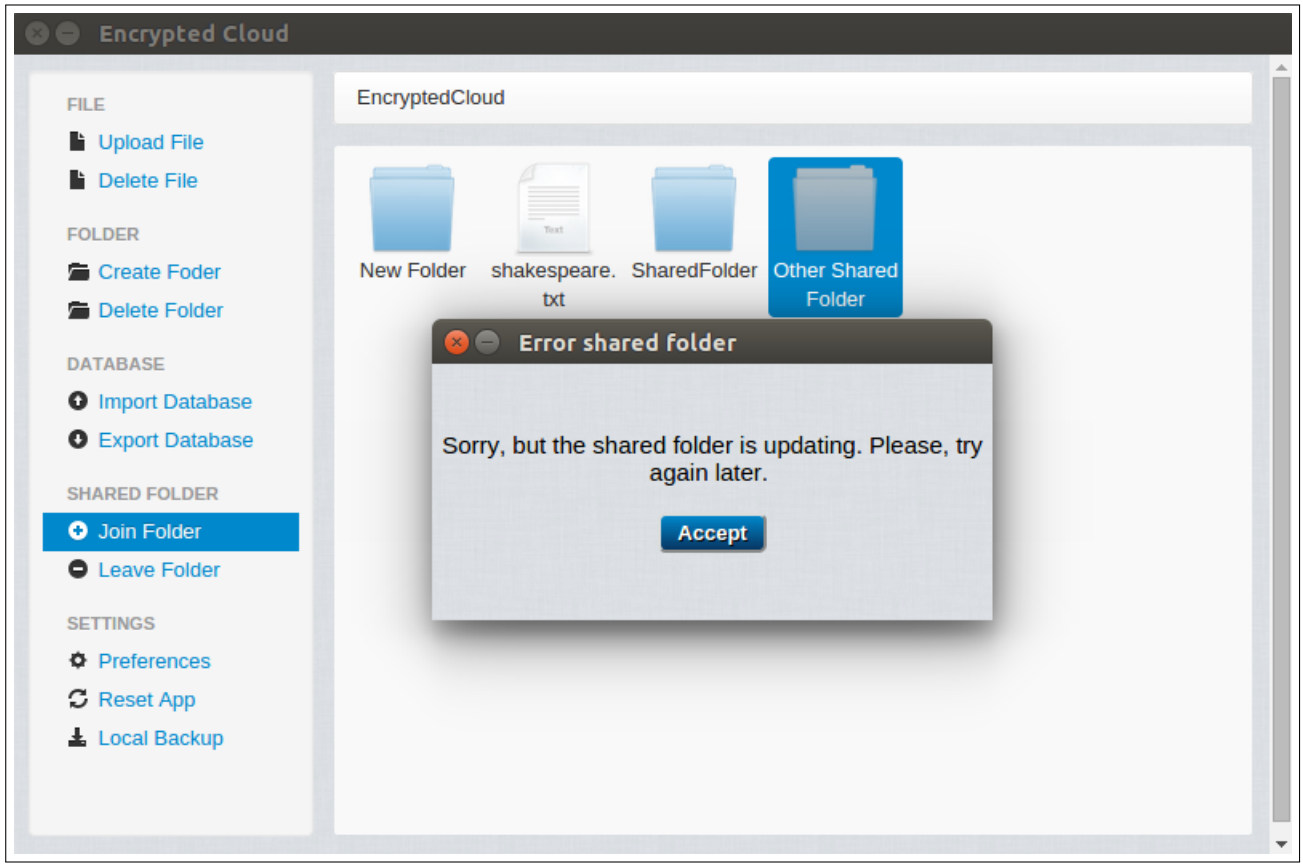


Figure 4.59: Leave folder integration test (II).

The following tests are made from two different approaches: first considering that the user who wants to leave the folder will be the owner, and on the other hand, from the perspective of a user who wants to leave a folder but she will not be the owner.

First, it is tested the case where the owner leaves a shared folder, "Other Shared Folder", verifying that she is notified and the elements of this shared folder are removed (Figure 4.60 and Figure 4.61).

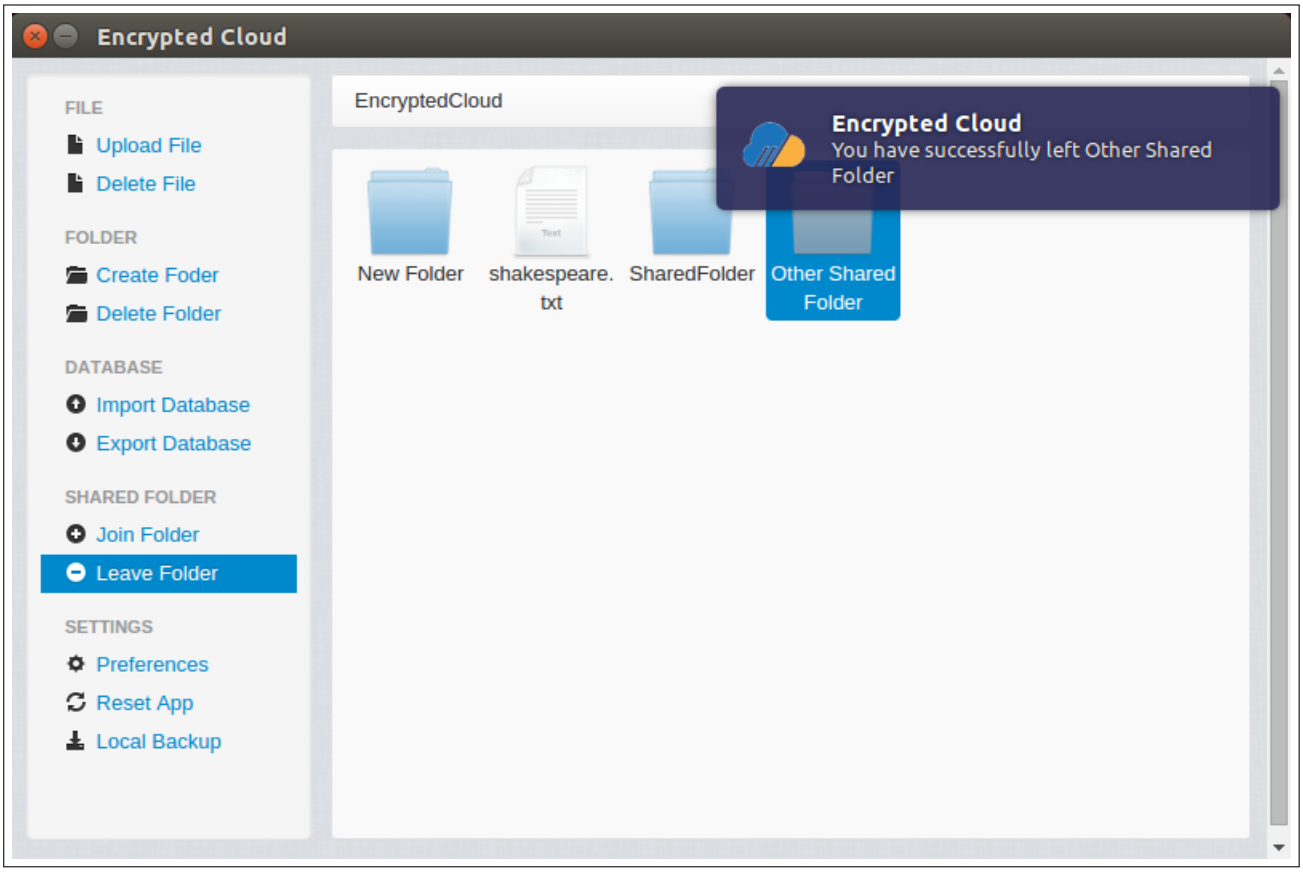


Figure 4.60: Leave folder integration test (III).

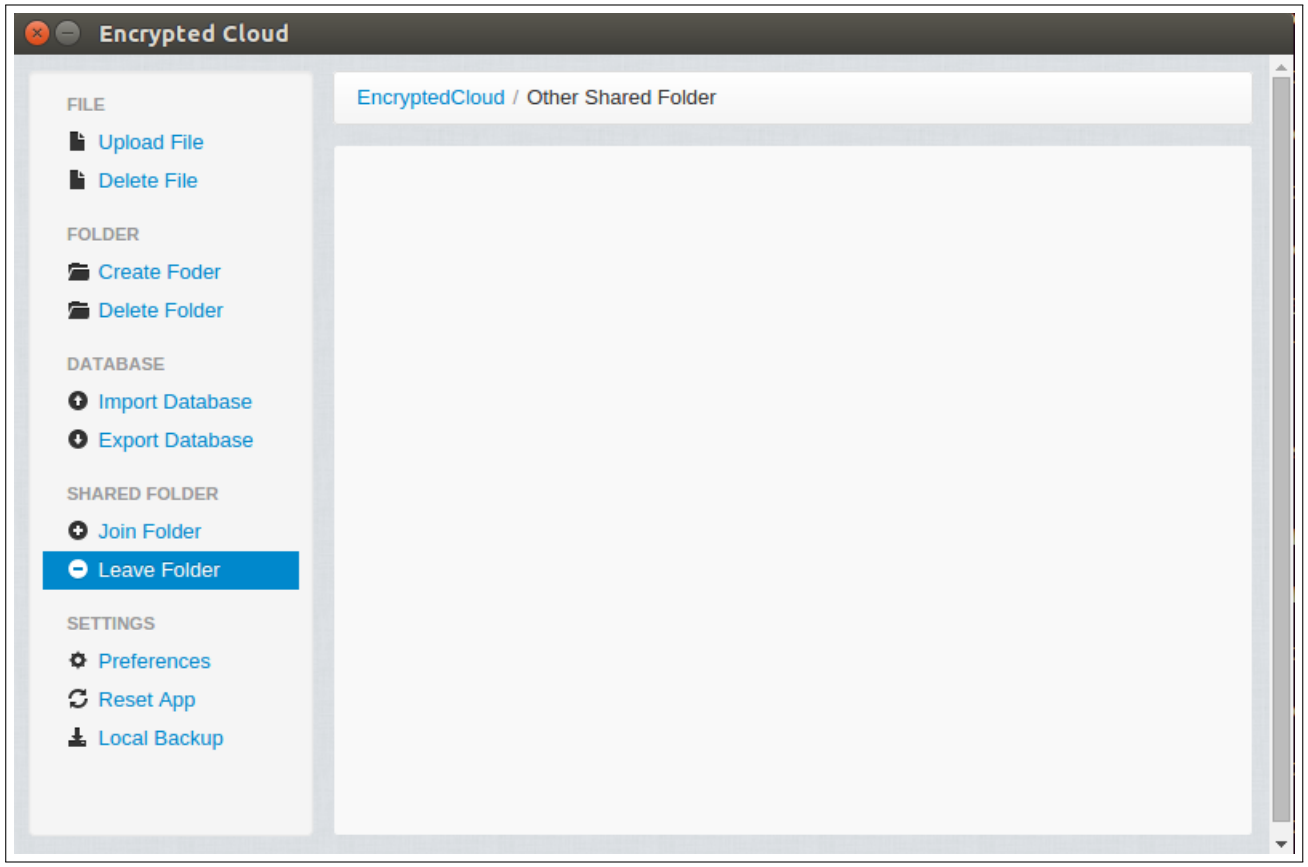


Figure 4.61: Leave folder integration test (IV).

Afterwards, the procedure performed in the previous test is repeated, in order to verify that a user cannot leave twice the same shared folder. In Figure 4.62 it is shown the notification which appears in this situation.

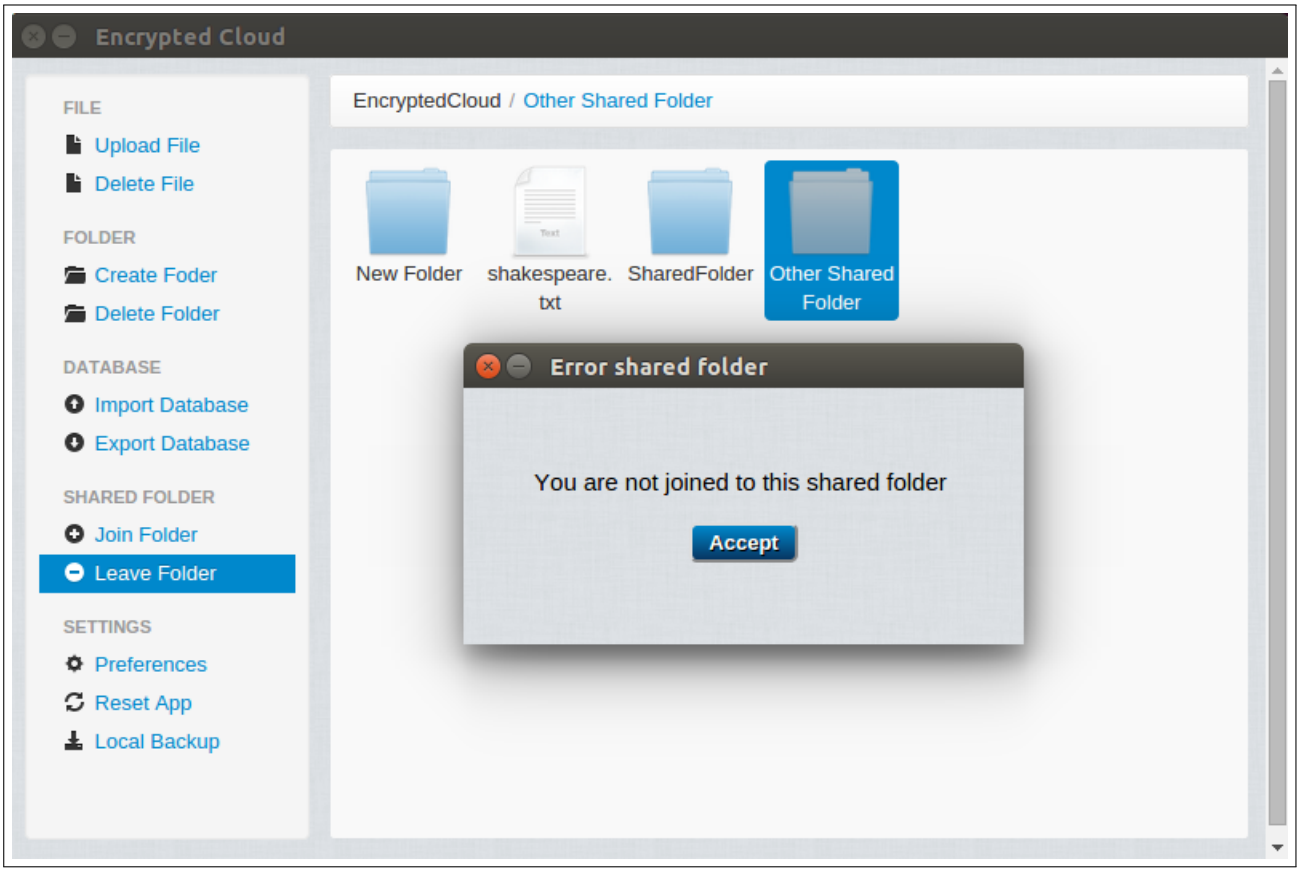


Figure 4.62: Leave folder integration test (V).

From the point of view of a user who is not the owner, when she wants to leave a shared folder, she will send a request to the owner, uploading it to a folder called *.leave_requests* contained in the shared folder she wants to leave. In order to test this behaviour, the user selects the "Shared Folder", to which a leave request is sent. This process can be seen in both Figure 4.63 and Figure 4.64.

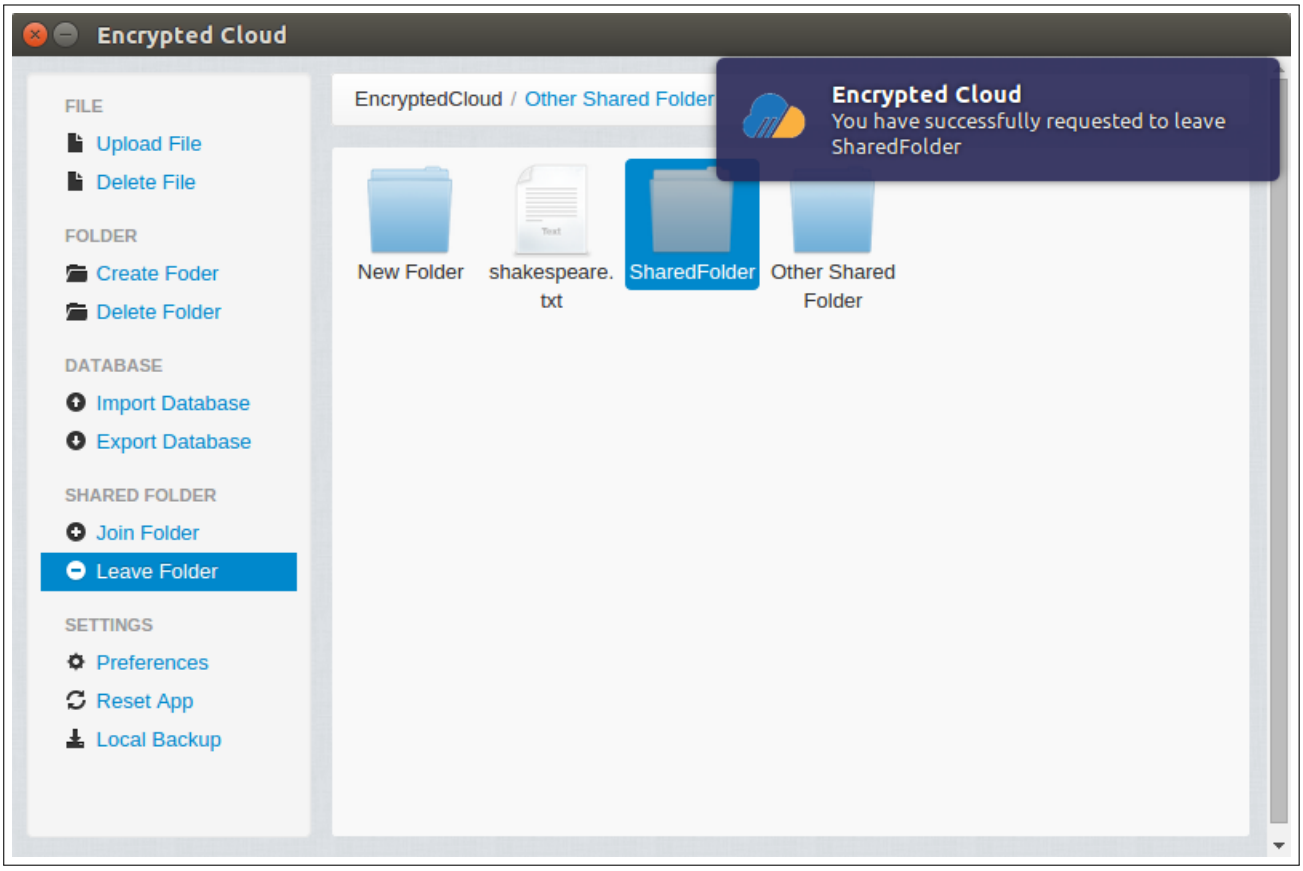


Figure 4.63: Leave folder integration test (VI).

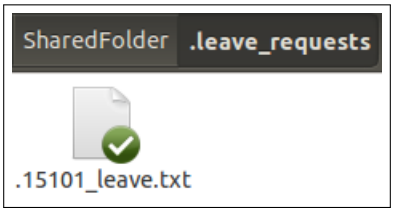


Figure 4.64: Leave folder integration test (VII).

Additionally, it is checked that if the request is deleted without being managed, it is uploaded again (Figure 4.65).

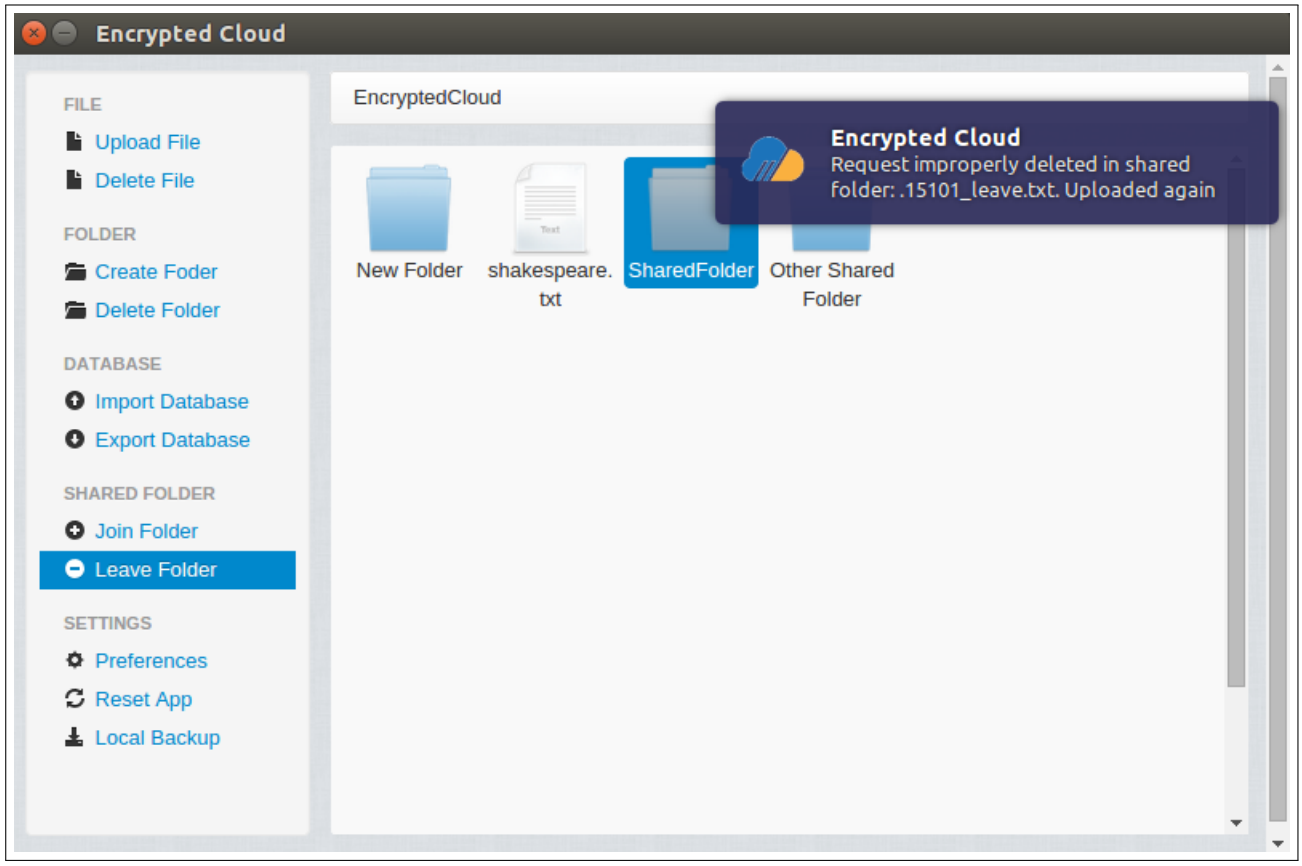


Figure 4.65: Leave folder integration test (VIII).

Chapter 5

Conclusion and future work

5.1 Conclusion of this work

In the current technological world, most companies use repository files to store their documents and, in most cases, they do not encrypt them. This practice is intensified by the use of FCS systems. Since information is a valuable element for both users and companies, any proposal that allows easily protect information stored in the cloud is of great importance. In this context, this work proposes Encrypted Cloud, a desktop application that represents a solution to meet this demand. Indeed, the user can encrypt her assets on the client side before being stored in the cloud provider, unifying all the free space provided by each cloud server. In addition, the application allows to share encrypted documents with other users, by incorporating a key distribution protocol into the cloud. Finally, under the premise that the cloud provider is reliable only with regard to the availability of assets that it stores, Encrypted Cloud performs a verification of the integrity of information assets as stored in the cloud.

The whole development carried out for this project has been reflected throughout this document. First, a brief introduction of cloud computing has been presented along with ten security challenges in cloud storage with their corresponding solutions. Afterwards, the solution implemented has been the main focus of this work. For this purpose, the full functionality of Encrypted Cloud has been described, explaining the steps that the user has to follow in order to use it. Finally, the test plan has been specified with the aim of verifying all the features that should meet the application.

Therefore, we can conclude that Encrypted Cloud meets with the requirements that were specified in the first stage of project. The final product has several improvements over existing similar solutions. First, it is very important the fact that Encrypted Cloud is an Open Source solution, and thus it allows to be evaluated by any user and/or potential developer. Furthermore, the application is designed and

implemented with a great independence between the security protocol and the resources storage system. As a consequence, the interface that supports the protocol for asset protection can be used in both on-line and off-line modes, allowing the use of different storage media (local hard drive, cloud storage servers, etc.). Moreover, all cryptographic and key management is performed on the client side, so the server has no cryptographic overload. Only with this policy it can be ensured that users have full control over their data. The server is only used to store the data, but not to preserve the confidentiality of these documents. Even in the case that the server modifies or destroys these data, the users could detect this malicious behaviour, since the protocol performs a continuous monitoring of files integrity. All these characteristics of Encrypted Cloud represent a high degree of innovation with respect to other available solutions in the current market.

From the personal point of view of the author of this master thesis, this project has provided a very meaningful framework to enhance the personal knowledge about information security. In specific, it has been possible to analyse thoroughly the main concerns about the security in cloud storage services. The proper achievement of the security and usability goals of this thesis has forced to study technologies as JavaScript and Node.js. These technologies are very important in these times due to the nowadays trend to implement cross-platform applications. Finally, we have in mind to write a paper to make known our application Encrypted Cloud to the scientific community.

5.2 Future work

Encrypted Cloud can be improved in several aspects, which could be covered in a future work. First, it might be interesting to recover files that have been uploaded and subsequently modified or eliminated by an unauthorised third party. Note that the efficiency of the project has not been evaluated. In a future work, this aspect could be improved by incorporating Merkle Trees [72], since the number of digital signatures could be reduced. On the other hand, the generation of the 2048-bit RSA key pair is computationally expensive. To mitigate this problem, it could be used elliptic curves [67]. Moreover, it could be a good improvement to have a server which manages all join and leave requests sent to shared folders, because the current version depends on the owner of each shared folder to carry out this task. This new server will be also in charge of registering the public keys of the users to share folders with. Encrypted Cloud has another limitation: it does not support external hard drives as a shared or non-shared directory, since these drives changed their drive letter, that is,

their absolute path, each time they are plugged into the computer. This a problem for Encrypted Cloud since it relies on absolute unchanged paths. Therefore, it seems necessary to search a mechanism with which assigns a static drive letter to an external drive [100, 101].

From the point of view of security information there are several features that can be improved. The first one is to encrypt the name of each asset, since an attacker could guess some of its contents only with its name. Besides, a new feature would be to check the strength of the user's password, making the user to provide a password longer than eight characters and with different character types. In this process of authentication it is also recommended to use 2SV mechanisms, so in the new version of Encrypted Cloud this feature will be available using a Node.js package called *speakeasy*[1]. Furthermore, in a future project, it could be defined different trust models according to different user profiles. For example, there will be users who want to manage their keys whereas others could prefer to use key servers because they trust their providers. Finally, a last goal could be to have document management systems on free cloud servers, which implies a version control of encrypted files. This last feature is an additional mechanism for preserving data integrity and a safeguard of information consistency. However, it should be noted the high complexity of version control, on which there are several proposals today. The problem of version control with encrypted data lies in the difficult reconciliation with the cryptographic diffusion principle, a problem which does not exist with stream cipher or block cipher in OFB or ECB mode [82]. Nevertheless, this type of encryption and these encryption modes may have security problems with image files, or files whose headers make known-plaintext attacks possible. Therefore, it should be considered that the documents are encrypted using AES in CBC mode. On this matter, we could take the *Rsyncrypto* project[2] as reference.

Regarding non-functional requirements, in the current version it is only allowed the modification of the user's password. Future work is intended to enable users to change the asset distribution protocol and the shared and non-shared directories, which were initially selected in the set up of Encrypted Cloud. Another important aspect could be to translate the application to other languages.

Finally, it could be a good option to take one or two months in order to formally test both the application security and usability. Security evaluation should perform with the help of automatic formal verification tools [90]. Correspondingly, the analysis of the application usability will be done with a meaningful number of users with

[1] www.npmjs.com/package/speakeasy
[2] www.rsyncrypto.lingnu.com/index.php/Home_Page

different technology knowledge. For this, it could be used different usability test methodologies which involve the user, like the well-known focus group and thinking aloud.

Appendix A

Gantt chart

ID	Task Name	Duration	Start
1	**Documentation**	**57 days**	**Fri 02/10/15**
2	Reading of related works	42 days	Fri 02/10/15
3	Study of similar solutions	15 days	Tue 01/12/15
4	**Analysis and Design**	**13 days**	**Tue 22/12/15**
5	Requirements analysis and application design	13 days	Tue 22/12/15
6	**Programming**	**61 days**	**Fri 08/01/16**
7	Study of desktop application technologies	10 days	Fri 08/01/16
8	MOOCs Node.js	14 days	Fri 22/01/16
9	Study of cryptographic APIs	8 days	Thu 11/02/16
10	Encryption examples with Node.js	4 days	Tue 23/02/16
11	Encryption of non-shared files in Encrypted Cloud	8 days	Mon 29/02/16
12	Encryption of shared files and key distribution in Encrypted Cloud	7 days	Thu 10/03/16
13	UI and other functionalities of Encrypted Cloud	10 days	Mon 21/03/16
14	**Tests**	**20 days**	**Mon 04/04/16**
15	Application tests	20 days	Mon 04/04/16
16	**Final report**	**22 days**	**Mon 02/05/16**
17	Final report writing	16 days	Mon 02/05/16
18	Review and correction of the final report	6 days	Tue 24/05/16

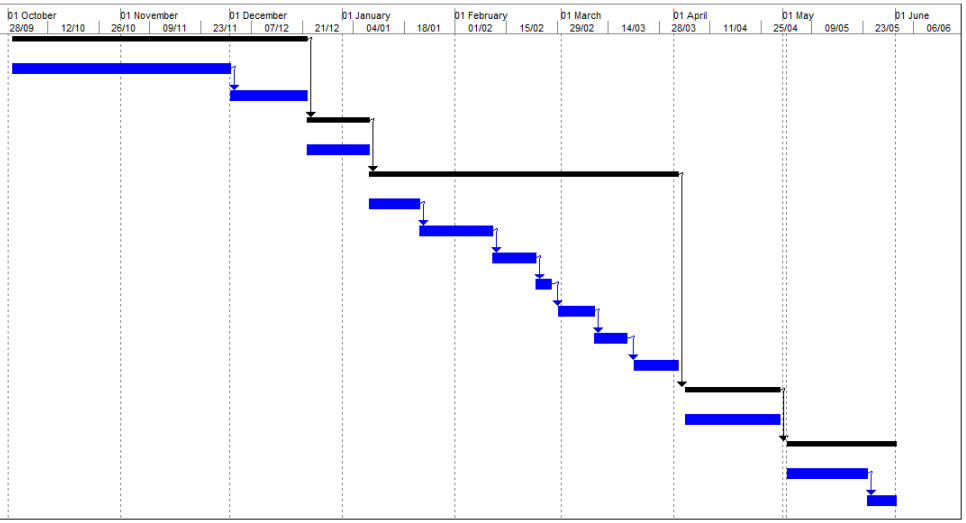


Figure A.1: Gantt Chart.

Appendix B

Unit tests results

```
Crypto Library
  #createRSA512KeyPairNotNull()
    ✓ should create RSA 512 bits key pair (234ms)
  #publicRSAKeyToString()
    ✓ should pass toString the RSA public key (45ms)
  #privateRSAKeyToString()
    ✓ should pass toString the RSA private key (88ms)
  #setGlobalRSAPublicKey()
    ✓ should set global variable RSA public key (45ms)
  #setGlobalRSAPrivateKey()
    ✓ should set global variable RSA private key (96ms)
  #createSeedAESKey256NotNull()
    ✓ should create AES 256 bits key
  #createIV16NotNull()
    ✓ should create IV 16 bytes
  #encryptDecryptAES256()
    ✓ should encrypt text plain and then decrypt it
  #createHashNotNull()
    ✓ should create hash from a string
```

Figure B.1: crypto_library.js unit tests results.

```
Database Library
  #checkDBConnectionOK()
    ✓ should check successful connection with database (814ms)
  #checkDBConnectionERROR()
    ✓ should check erroneous connection with database (222ms)
  #checkUpdateDBPassword()
    ✓ should check successful database password update (1035ms)
  #checkInsertEncryptedCloud()
    ✓ should check a register is inserted into EncryptedCloud entity (852ms)
  #checkInsertAsset()
    ✓ should check a register is inserted into Asset entity (718ms)
  #checkInsertRequest()
    ✓ should check a register is inserted into SharedFolderRequest entity (566ms)
  #checkInsertSharedFolderUsers()
    ✓ should check a register is inserted into SharedFolderUsers entity (610ms)
  #checkDeleteAsset()
    ✓ should check a register is deleted from Asset entity (510ms)
  #checkDeleteRequest()
    ✓ should check a register is deleted from SharedFolderRequest entity (381ms)
  #checkGetFilesUploadedFromDB()
    ✓ should check a register is read from Asset entity (336ms)
  #checkGetNonSharedAssetsFromDB()
    ✓ should check a non-shared asset is read from Asset entity (409ms)
  #checkReadRequestsFromDB()
    ✓ should check a request is read from SharedFolderRequest entity (433ms)
  #checkUpdateUsersSharedFolder()
    ✓ should check a register is updated in SharedFolderUsers entity (475ms)
  #checkGetUsersSharedFolder()
    ✓ should check a register is read from SharedFolderUsers entity (250ms)
  #checkDeleteUsersSharedFolder()
    ✓ should check a register is deleted from SharedFolderUsers entity (421ms)
  #checkUpdateAssetProtocol()
    ✓ should check a register is updated into EncryptedCloud entity (363ms)
```

Figure B.2: database_library.js unit tests results.

Figure B.3: fs_library.js and shared_folder_library.js unit tests results.

Appendix C

Integration tests results

C.1 Check set up form integration test

When the user starts Encrypted Cloud for the first time, it shows an initial form with the application settings. When the user clicks on the "Start app" button, the fields filled in by the user are validated.

The first test checks that none of the password fields are empty (Figure C.1 and Figure C.2).

Figure C.1: Check set up integration test (I).

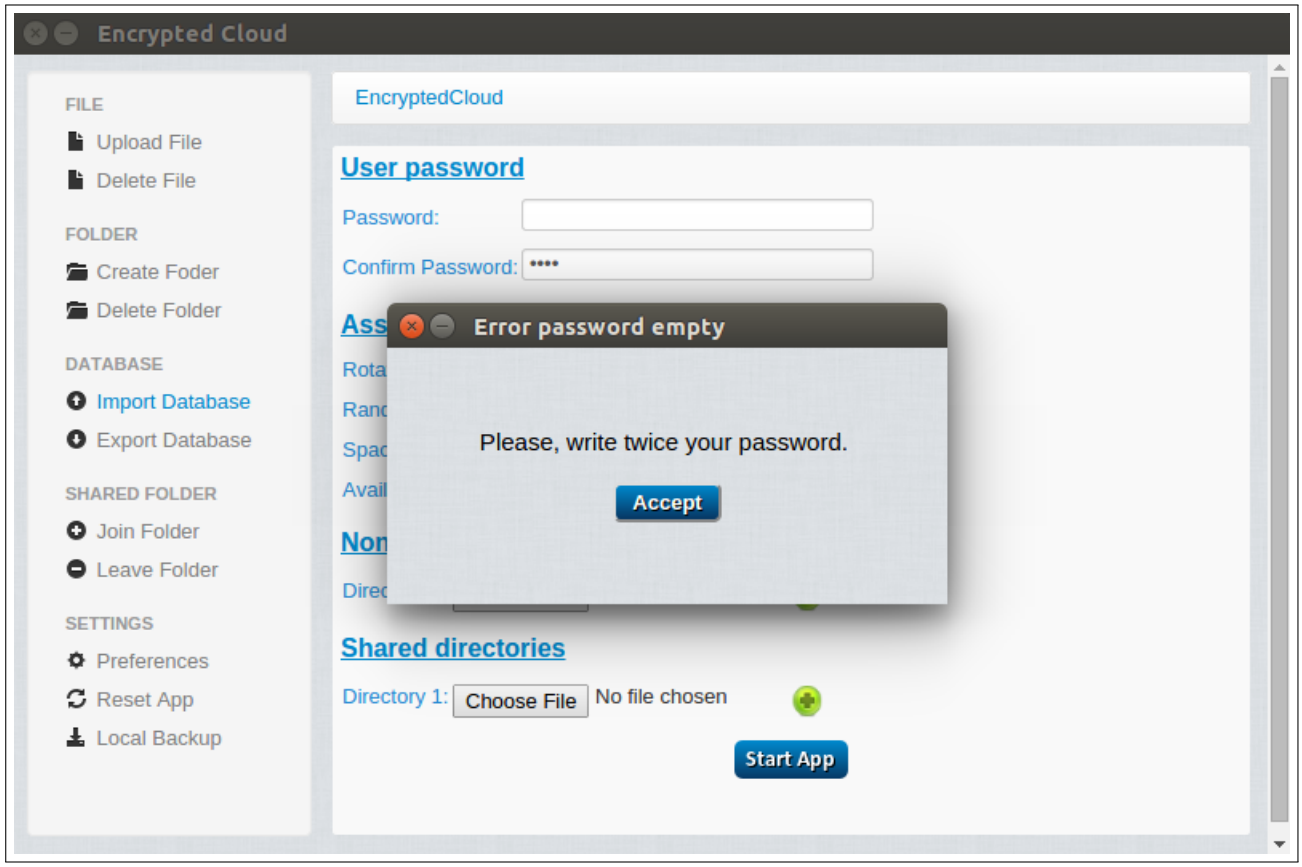


Figure C.2: Check set up integration test (II).

The second test verifies that both passwords typed by the user are equal:

Figure C.3: Check set up integration test (III).

Finally, if there is no problem with password verification, it is checked if the user has selected at least one non-shared directory:

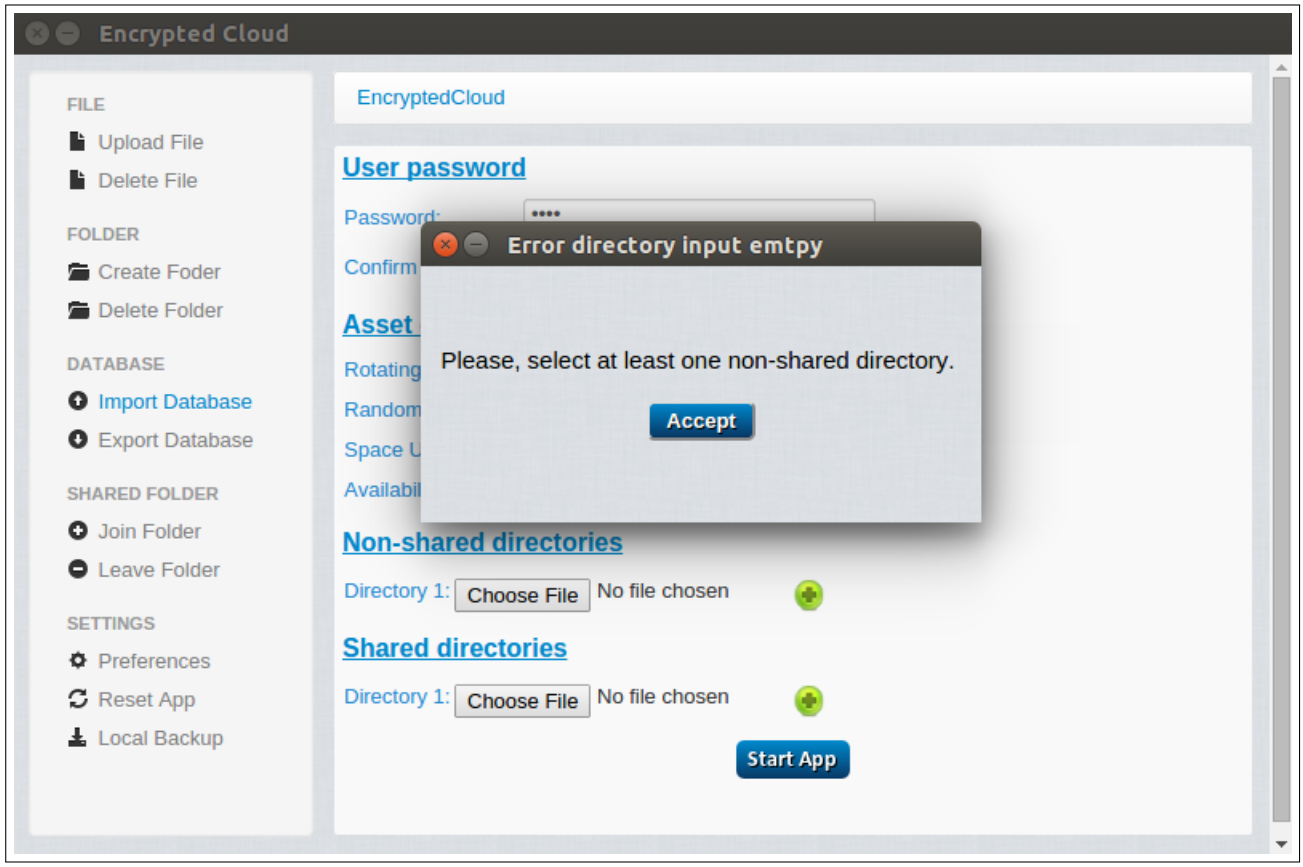


Figure C.4: Check set up integration test (IV).

C.2 Login integration test

If the user has previously made the set up, every time she starts the application she must provide the password to be authenticated. This login activity checks if the password submitted by the user enables the connection with the encrypted database.

In this test, the user was signed up with "alex" as password. Then, it is verified that if a different password is introduced, the user is notified that her password is incorrect or the database is corrupted, as shown in Figure C.5.

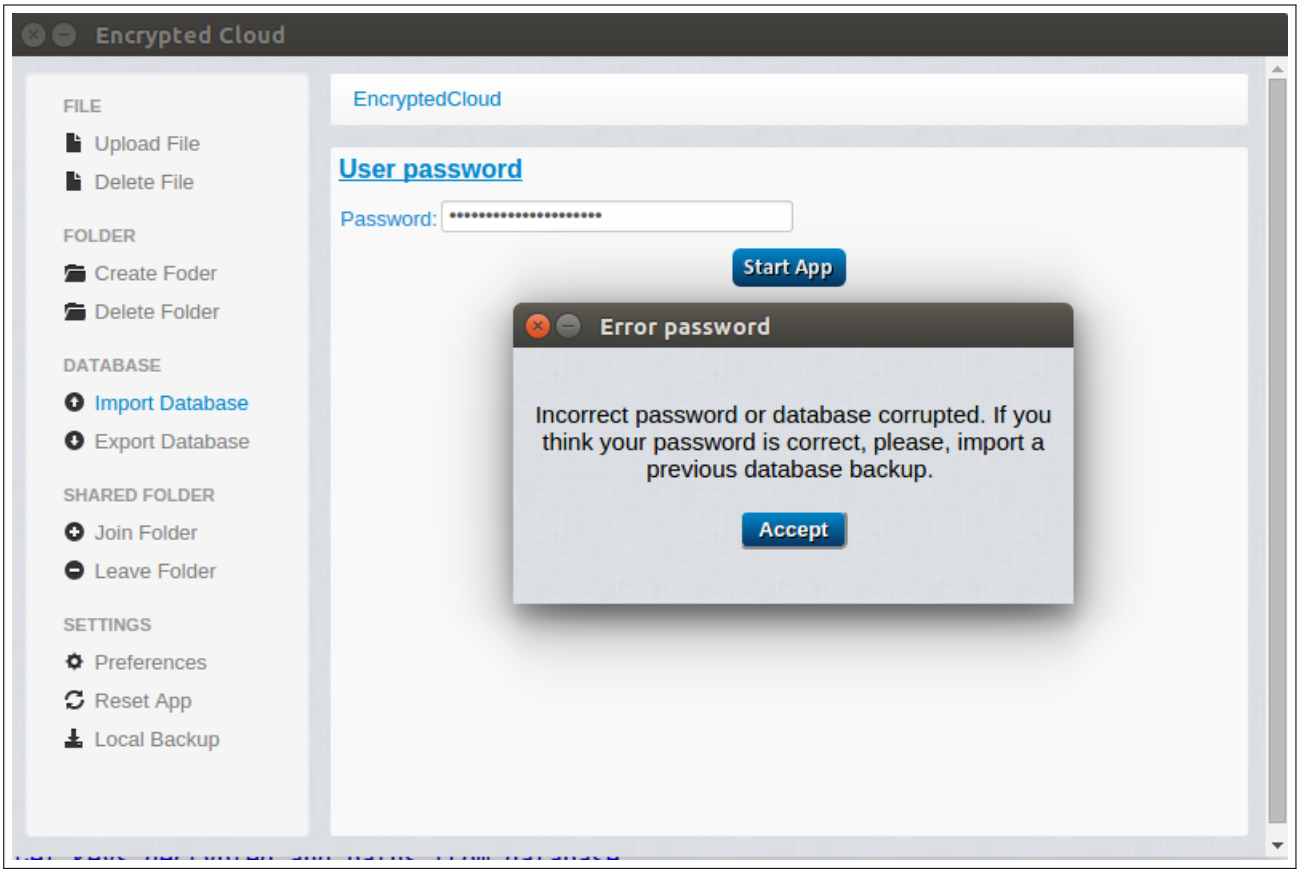


Figure C.5: Login integration test (I).

On the other hand, if the password is correct, the user can access the file explorer of the application (Figure C.6 and Figure C.7).

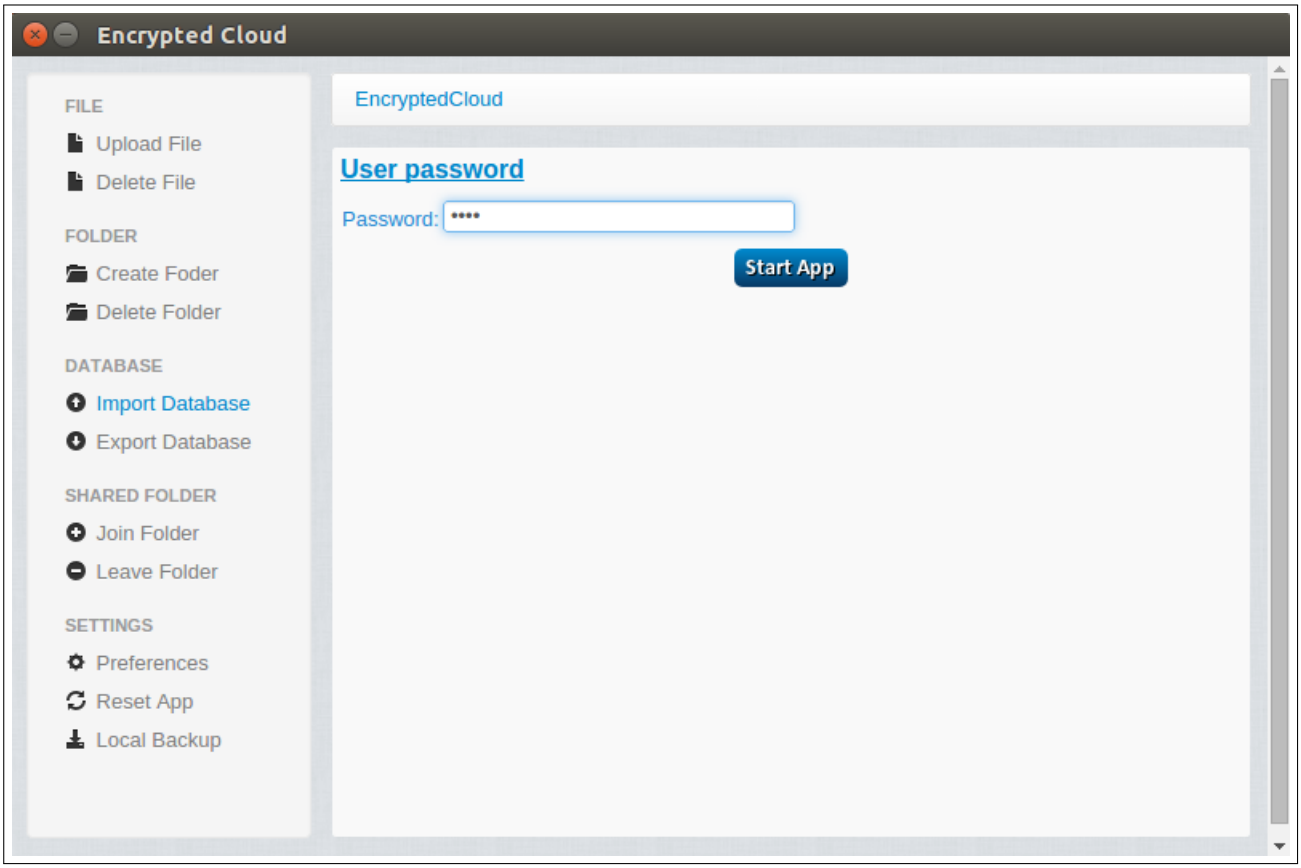


Figure C.6: Login integration test (II).

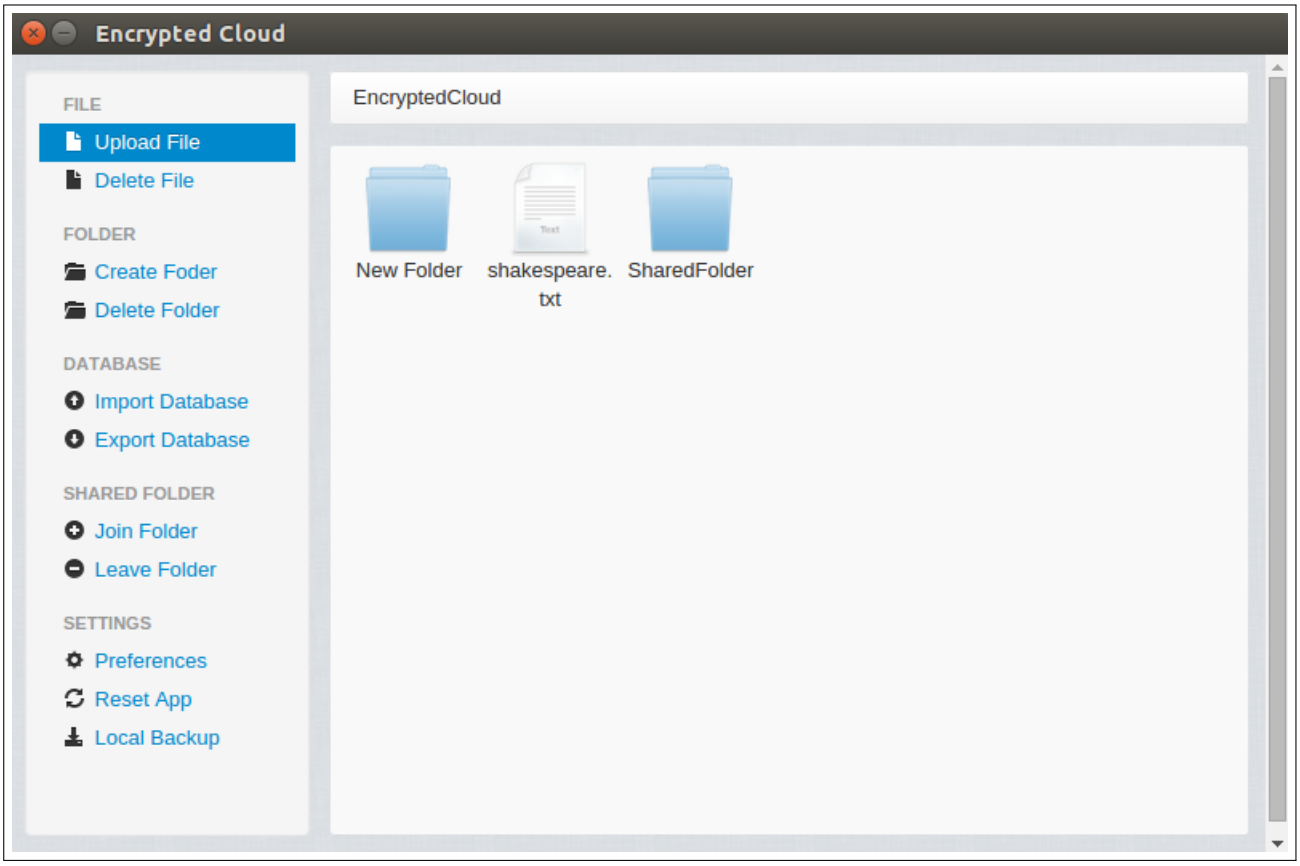


Figure C.7: Login integration test (III).

C.3 Asset distribution protocol integration test

This subsection tests three different functionalities: the asset distribution protocol and both the creation and deletion of folders.

For the first test, the user selects three non-shared folders and "Rotating" as the asset distribution protocol:

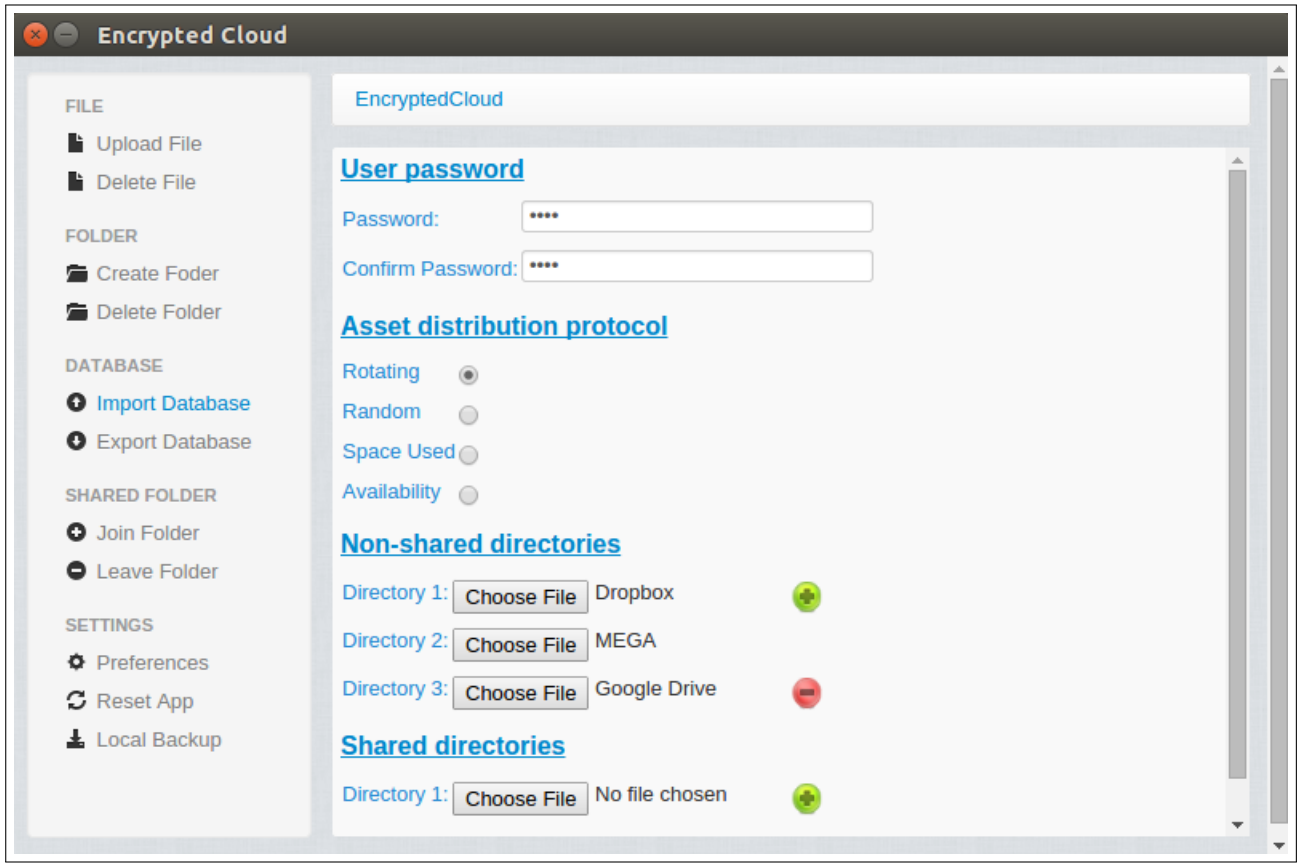


Figure C.8: Asset distribution protocol integration test (I).

From the application root path, the user creates three folders: "Folder1", "Folder2", and "Folder3". For this, the user has to click on the link "Create Folder" and then provide the name of each folder:

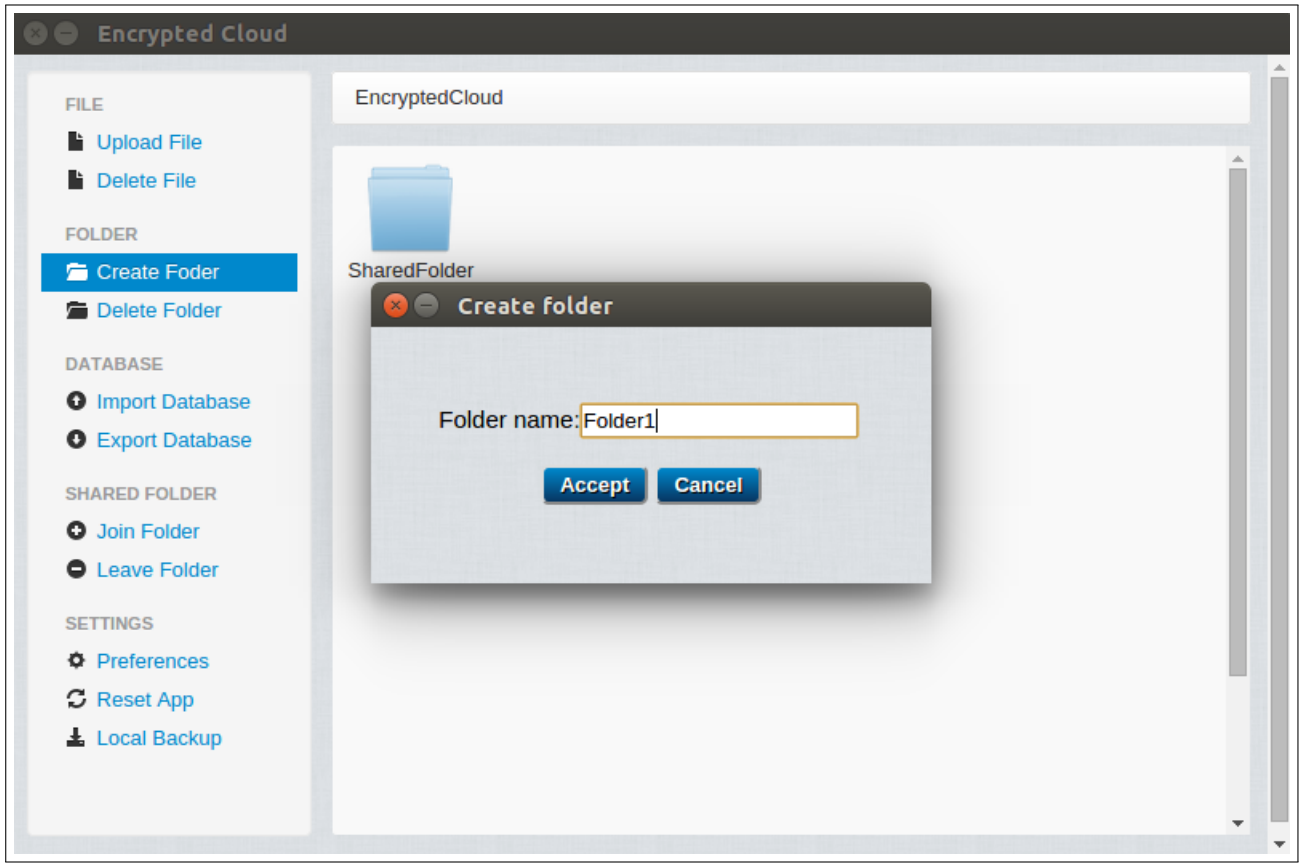


Figure C.9: Asset distribution protocol integration test (II).

It is verified that each folder is created in a different non-shared folder, according to the "Rotating" protocol:

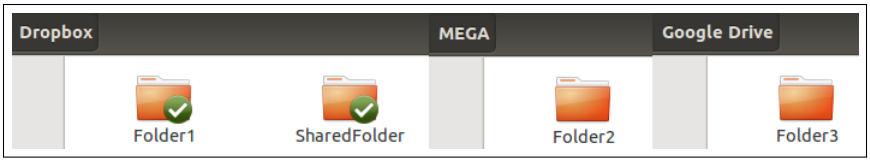


Figure C.10: Asset distribution protocol integration test (III).

Finally, the "Folder1" is removed. This being the case, the user selects this folder and clicks on "Delete Folder":

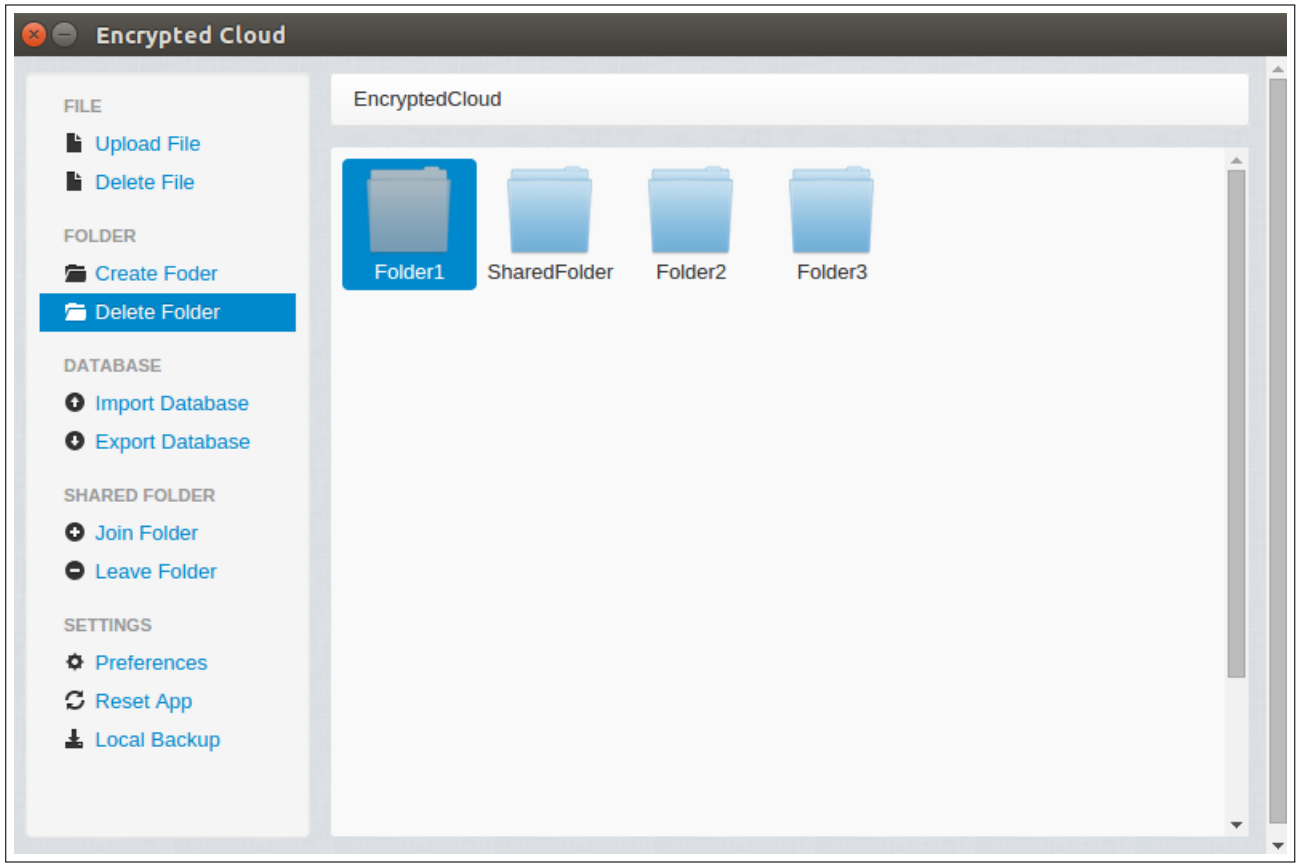


Figure C.11: Asset distribution protocol integration test (IV).

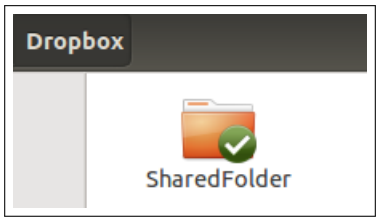


Figure C.12: Asset distribution protocol integration test (V).

In the second test, the user selects the same non-shared folders as in the previous one, but with "Random" as the asset distribution protocol:

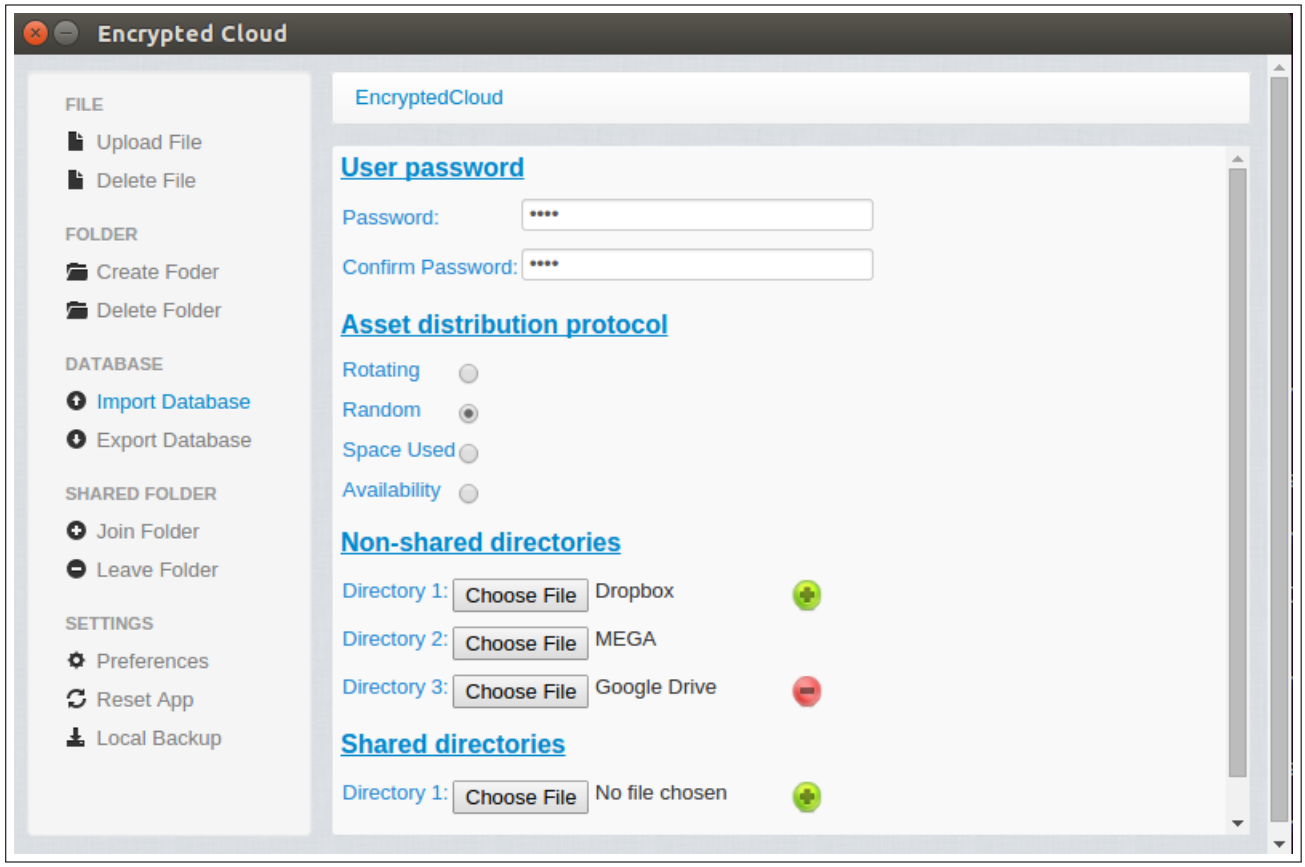


Figure C.13: Asset distribution protocol integration test (VI).

The user creates three folders ("Folder1", "Folder2", and "Folder3") again, but in this case, two of them are stored randomly in the non-shared directory "Google Drive":

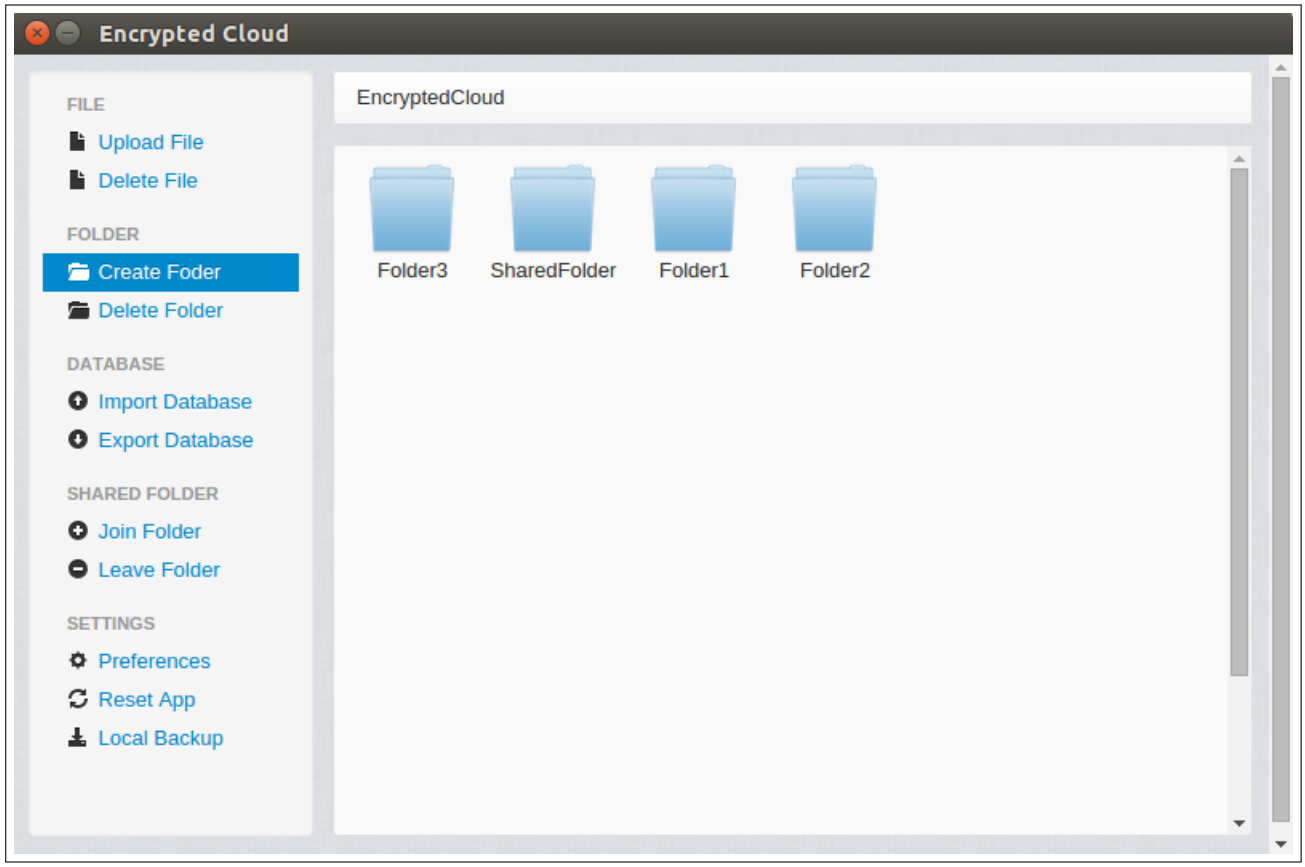


Figure C.14: Asset distribution protocol integration test (VII).

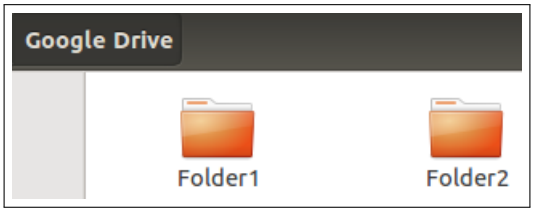


Figure C.15: Asset distribution protocol integration test (VIII).

For the third test, the user selects the local folders “Dropbox” and “Google Drive” as the non-shared directories and “Space Used” as the asset distribution protocol:

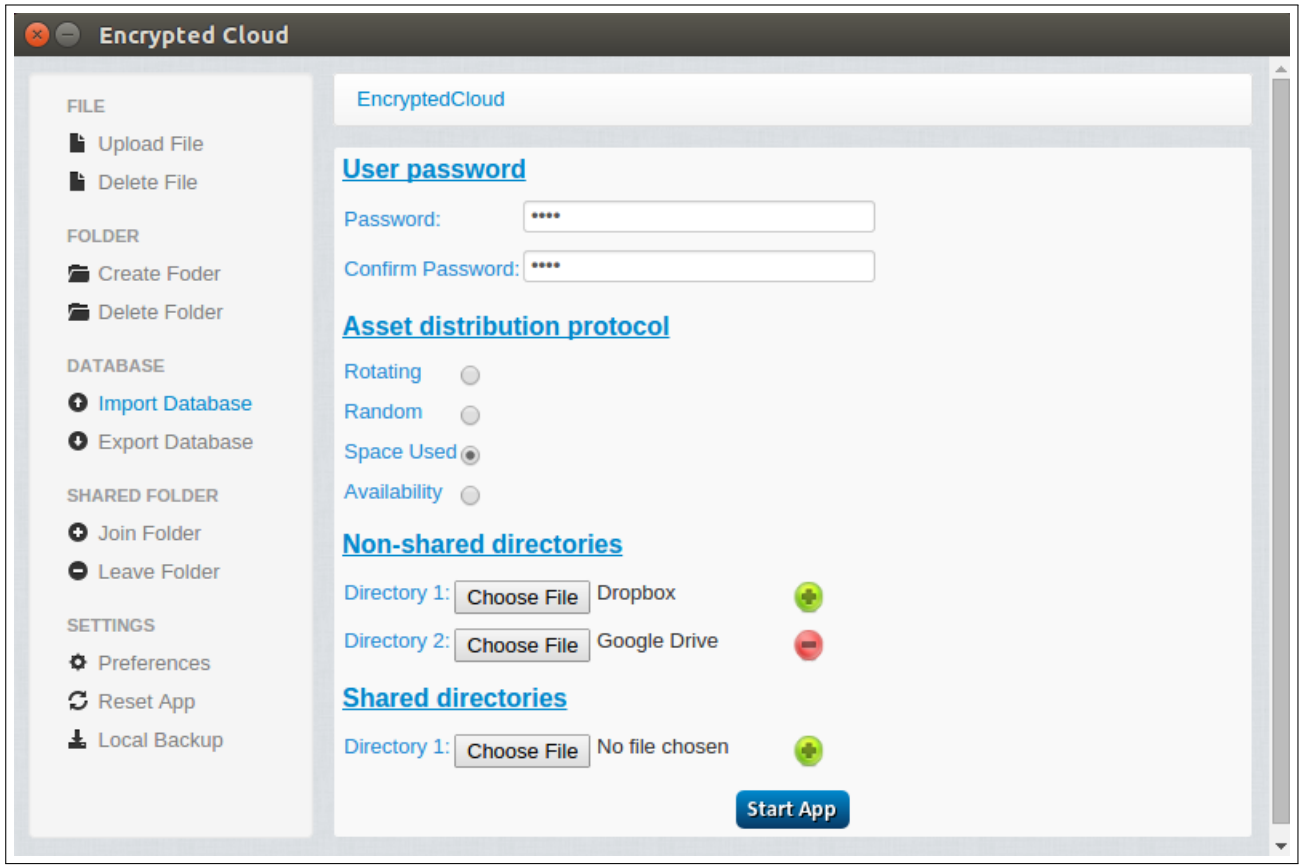


Figure C.16: Asset distribution protocol integration test (IX).

Note that "Dropbox" has an internal folder (Figure C.17) while "Google Drive" is empty (Figure C.18).

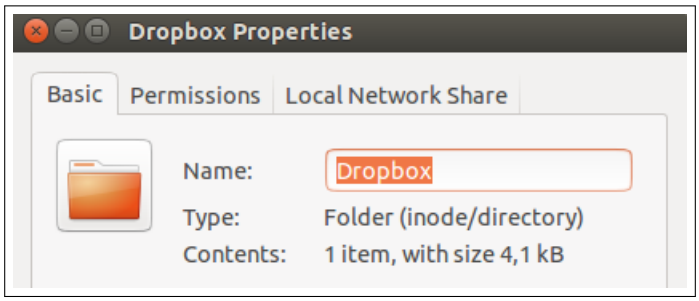


Figure C.17: Asset distribution protocol integration test (X).

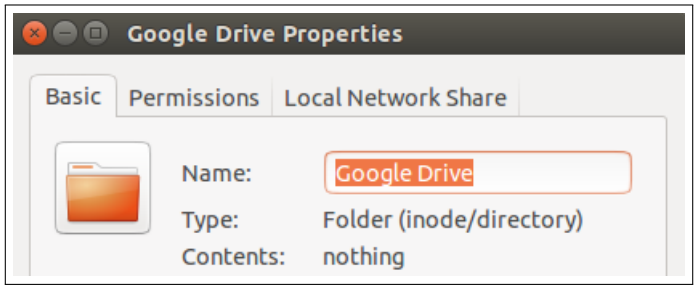


Figure C.18: Asset distribution protocol integration test (XI)

Therefore, if a new folder is created with this protocol, it will be stored in "Google Drive", as shown in Figure C.19 and in Figure C.20.

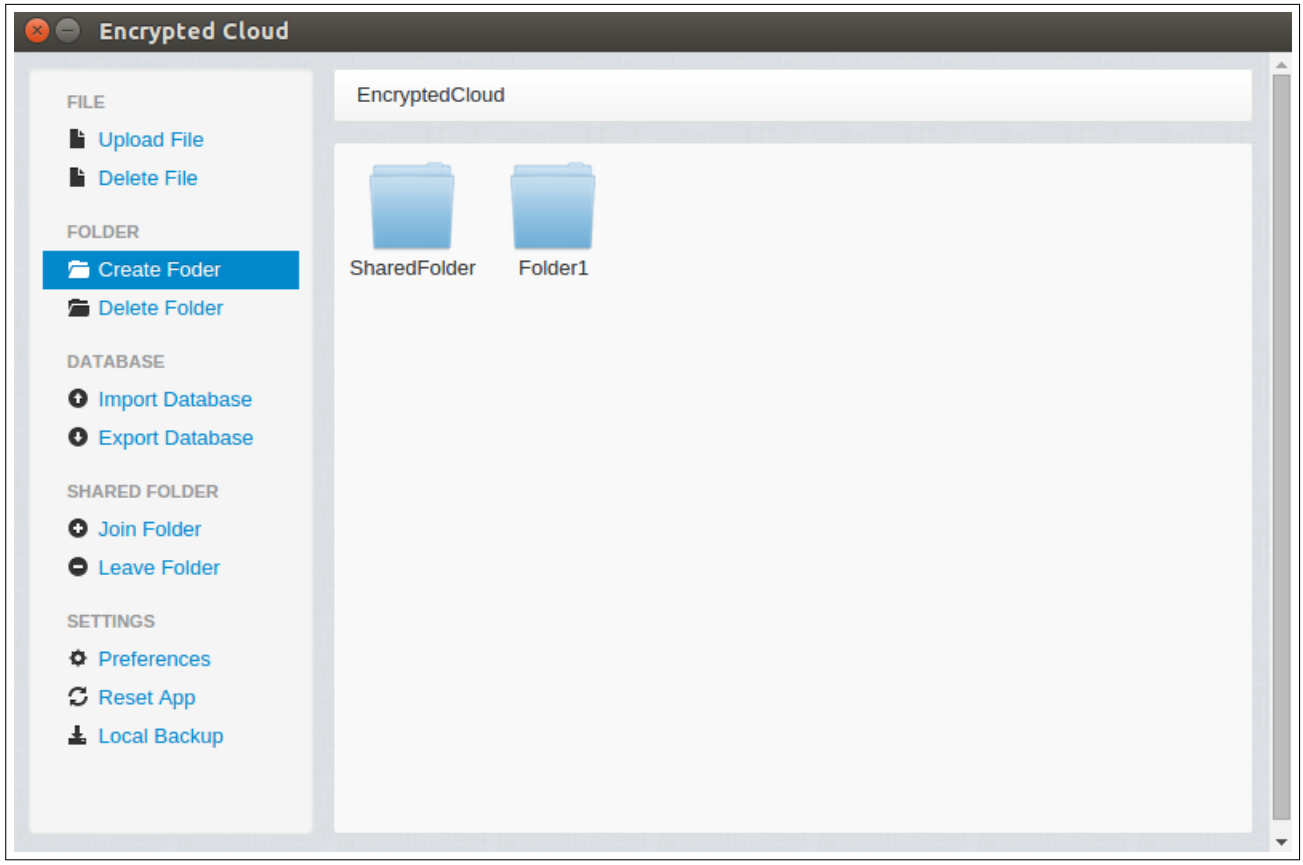


Figure C.19: Asset distribution protocol integration test (XII).

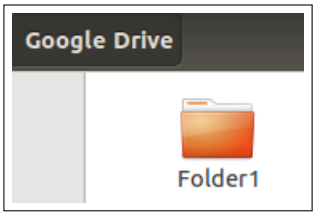


Figure C.20: Asset distribution protocol integration test (XIII).

Finally, it is tested the "Availability" protocol with the following non-shared directories:

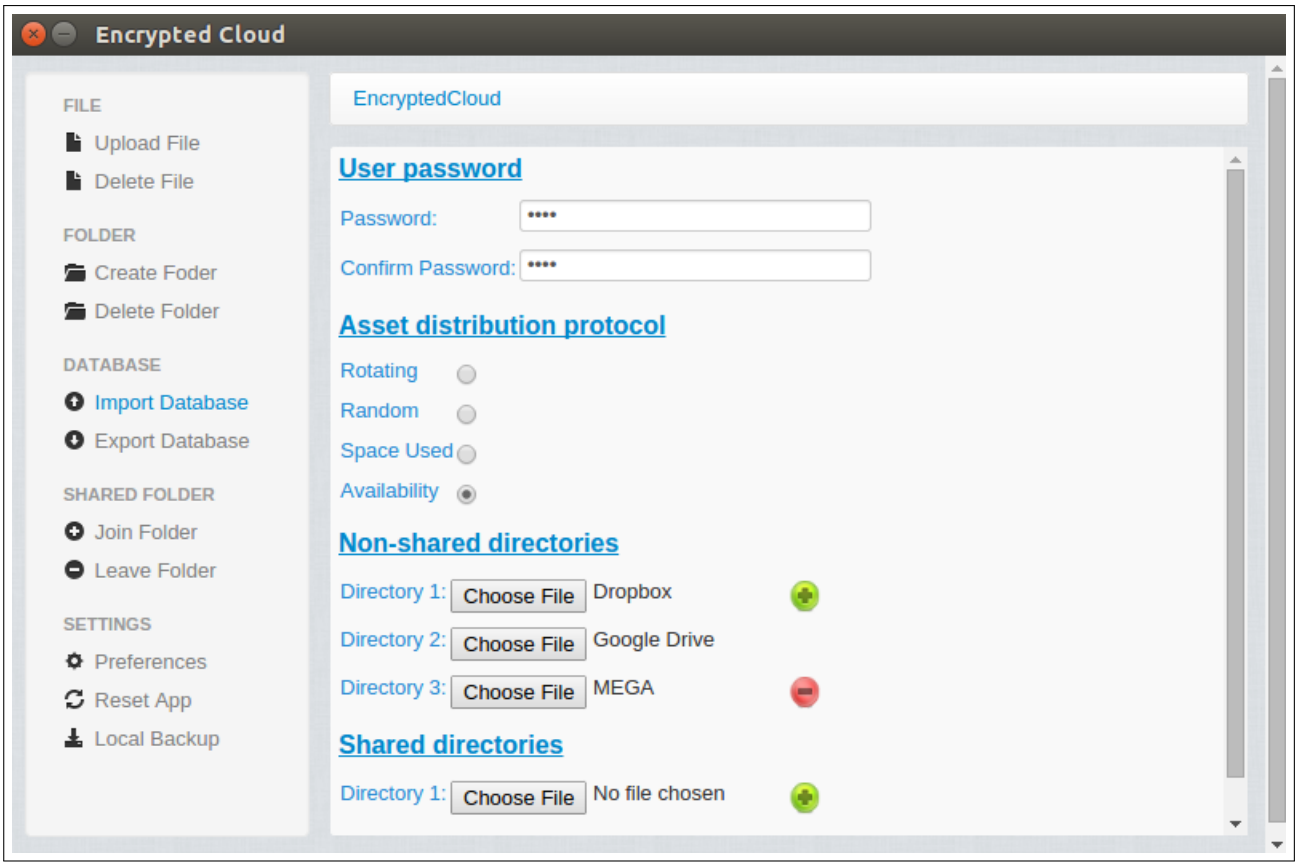


Figure C.21: Asset distribution protocol integration test (XIV).

Afterwards, the "Folder1" is created, and it is verified that is stored in the three non-shared directories:

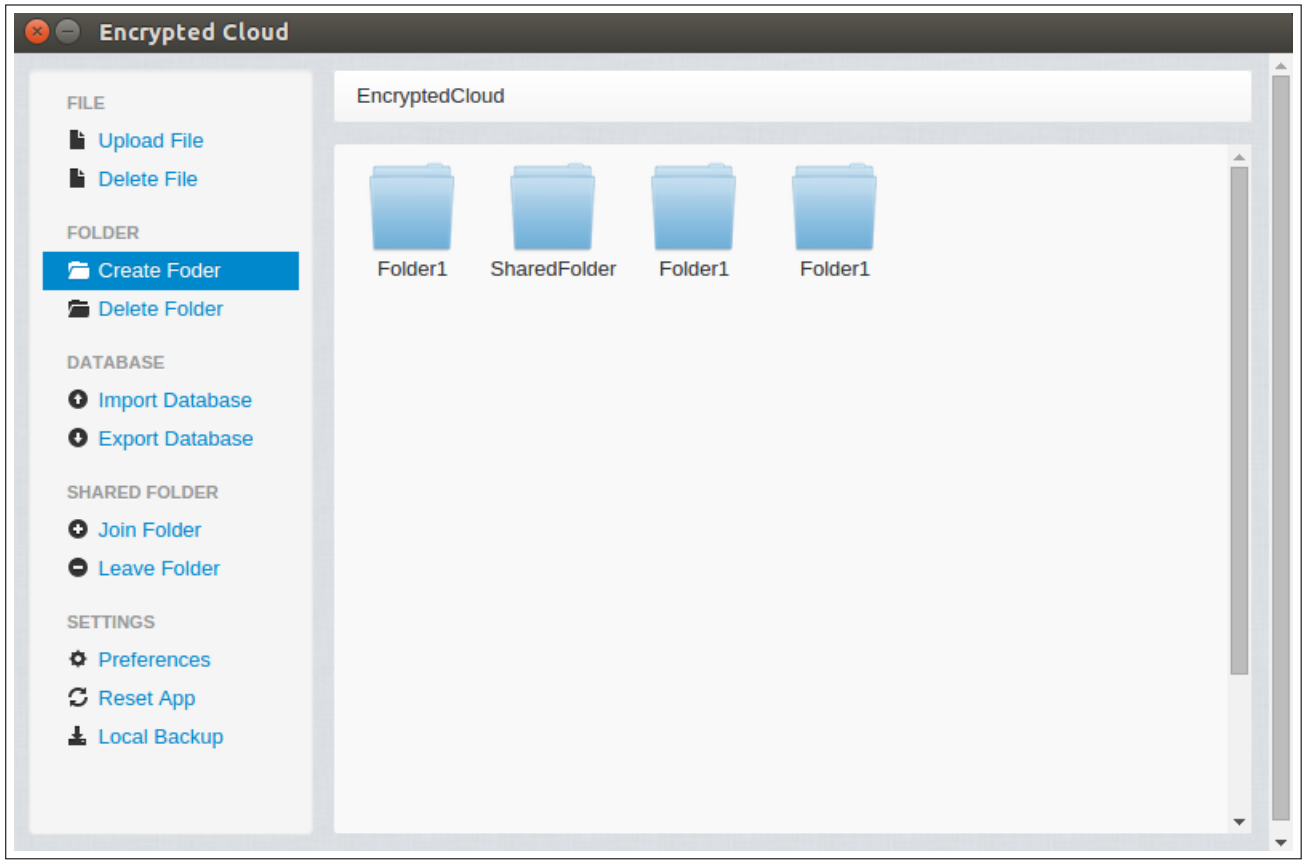


Figure C.22: Asset distribution protocol integration test (XV).

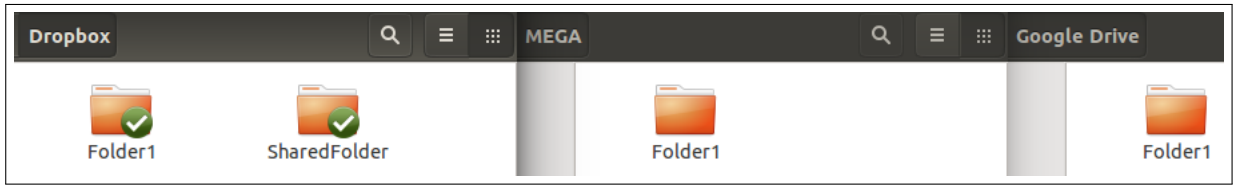


Figure C.23: Asset distribution protocol integration test (XVI).

In addition, if the user would decide to delete one of the three folders entitled "Folder1", its other copies would be also removed. These tests could also have done with files instead of folders.

C.4 Password modification integration test

Encrypted Cloud provides the functionality to change the user's password through the link "Preferences" on the left side panel. In this test, the user's password is

“alex”:

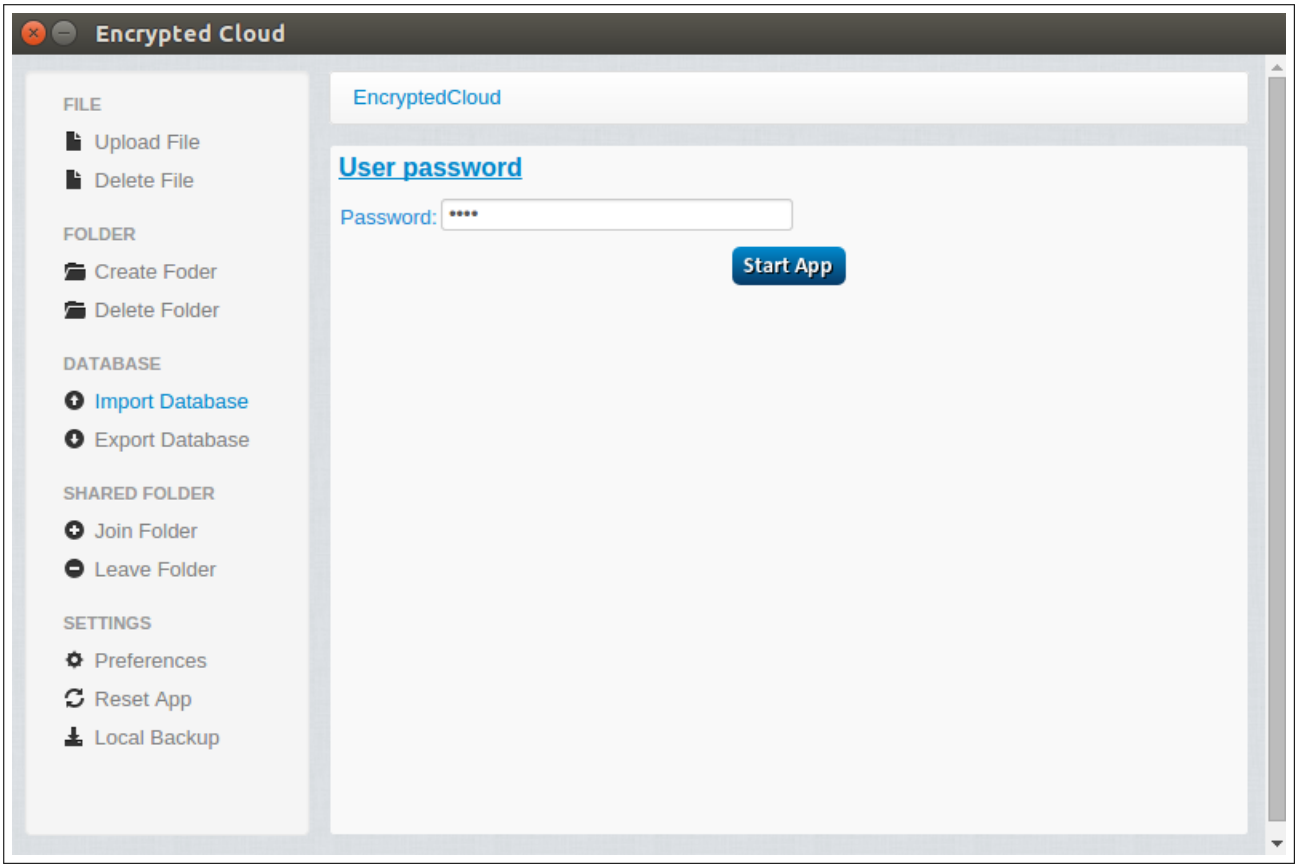


Figure C.24: Password modification integration test (I).

This user has uploaded a file called “shakespeare.txt” that she can display:

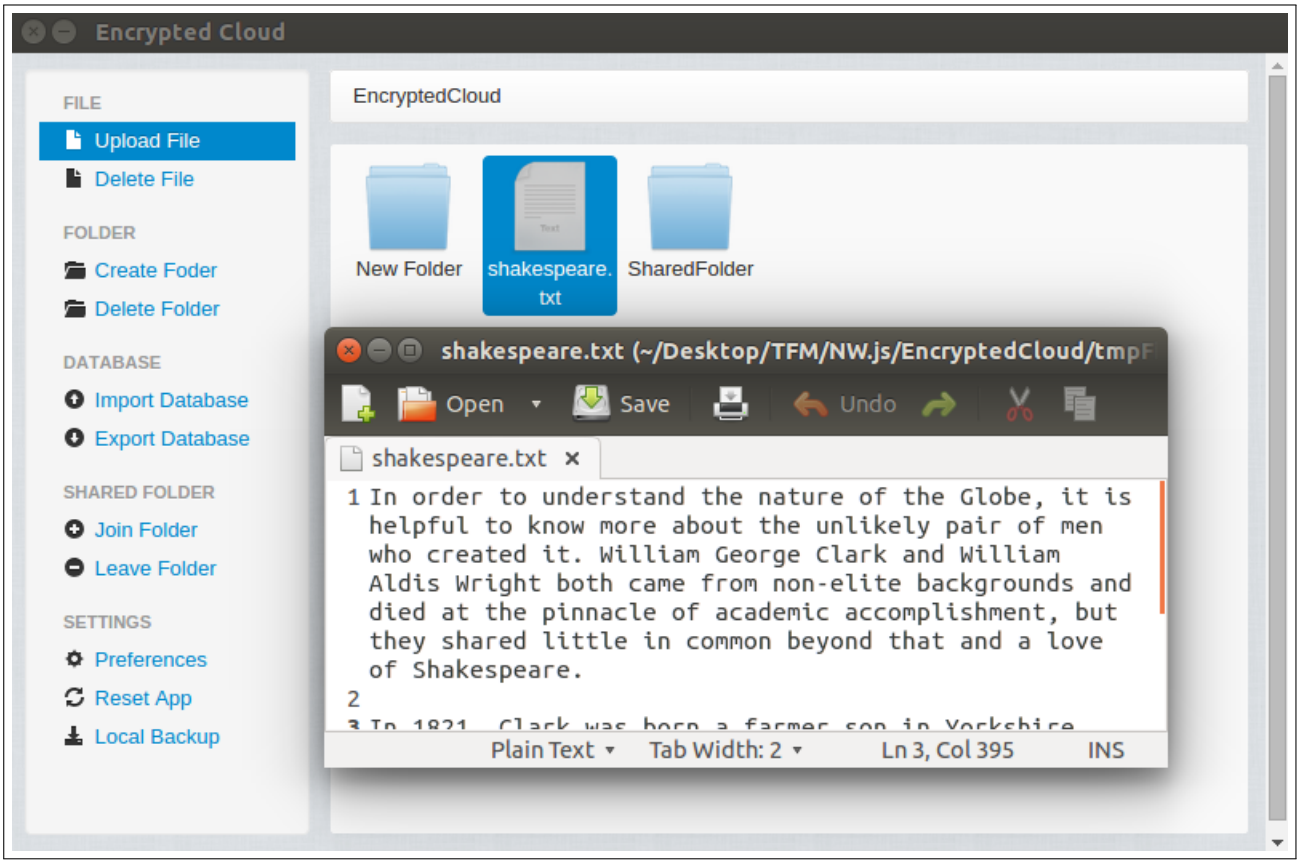


Figure C.25: Password modification integration test (II).

Then, the user decides to change her password, so she clicks on "Preferences" and the following pop-up is shown:

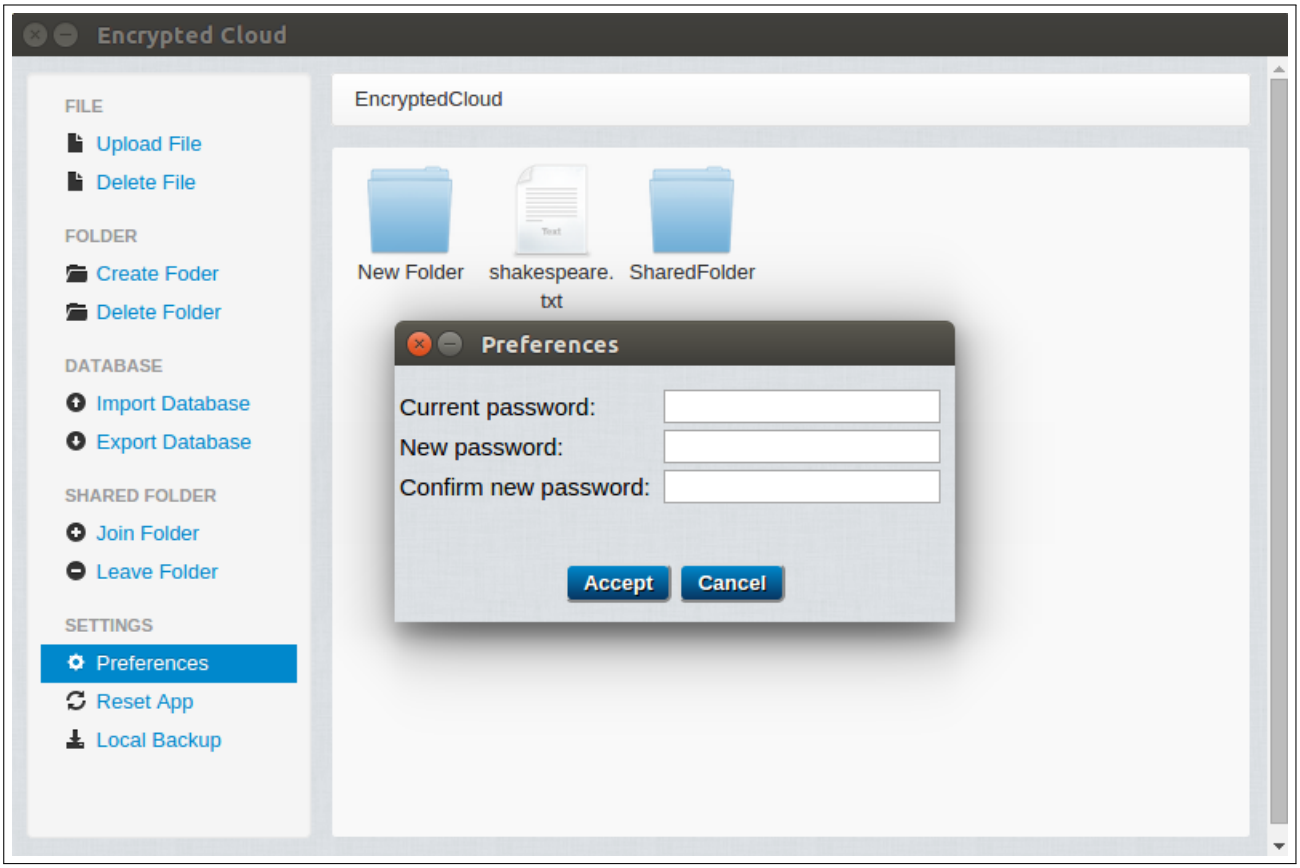


Figure C.26: Password modification integration test (III).

In this pop-up is verified that (I) no field is empty; (II) the current password is correct; (III) both new passwords are identical (Figure C.27, Figure C.28, and Figure C.29).

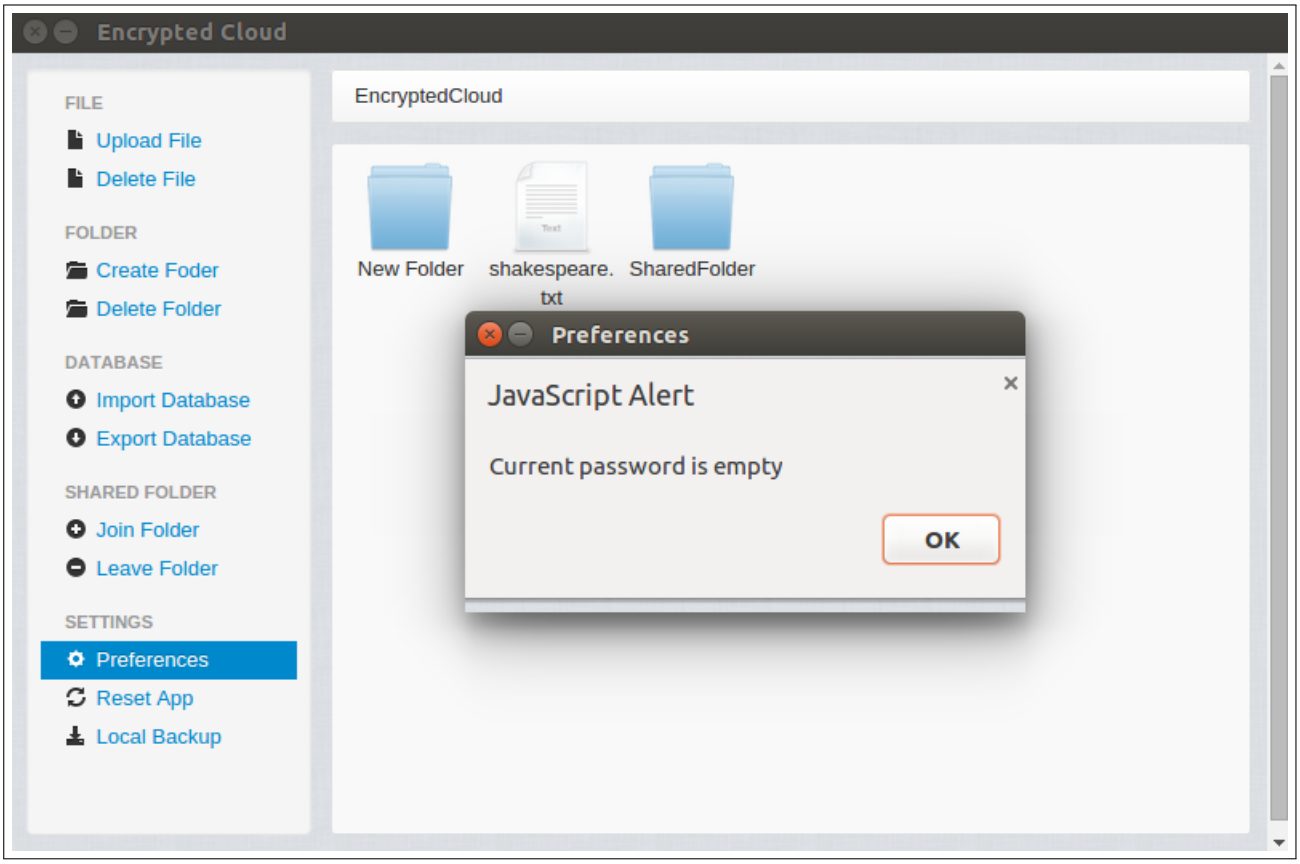


Figure C.27: Password modification integration test (IV).

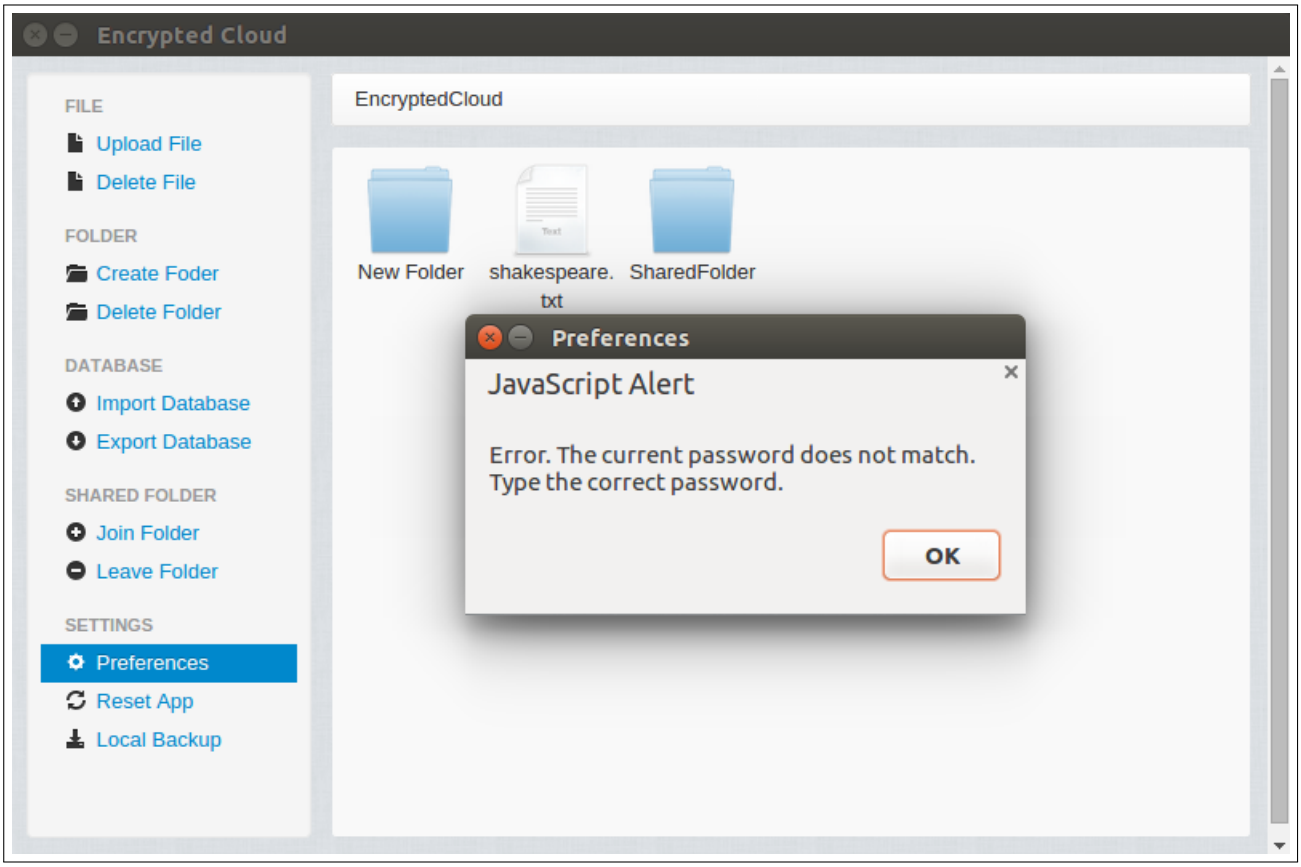


Figure C.28: Password modification integration test (V).

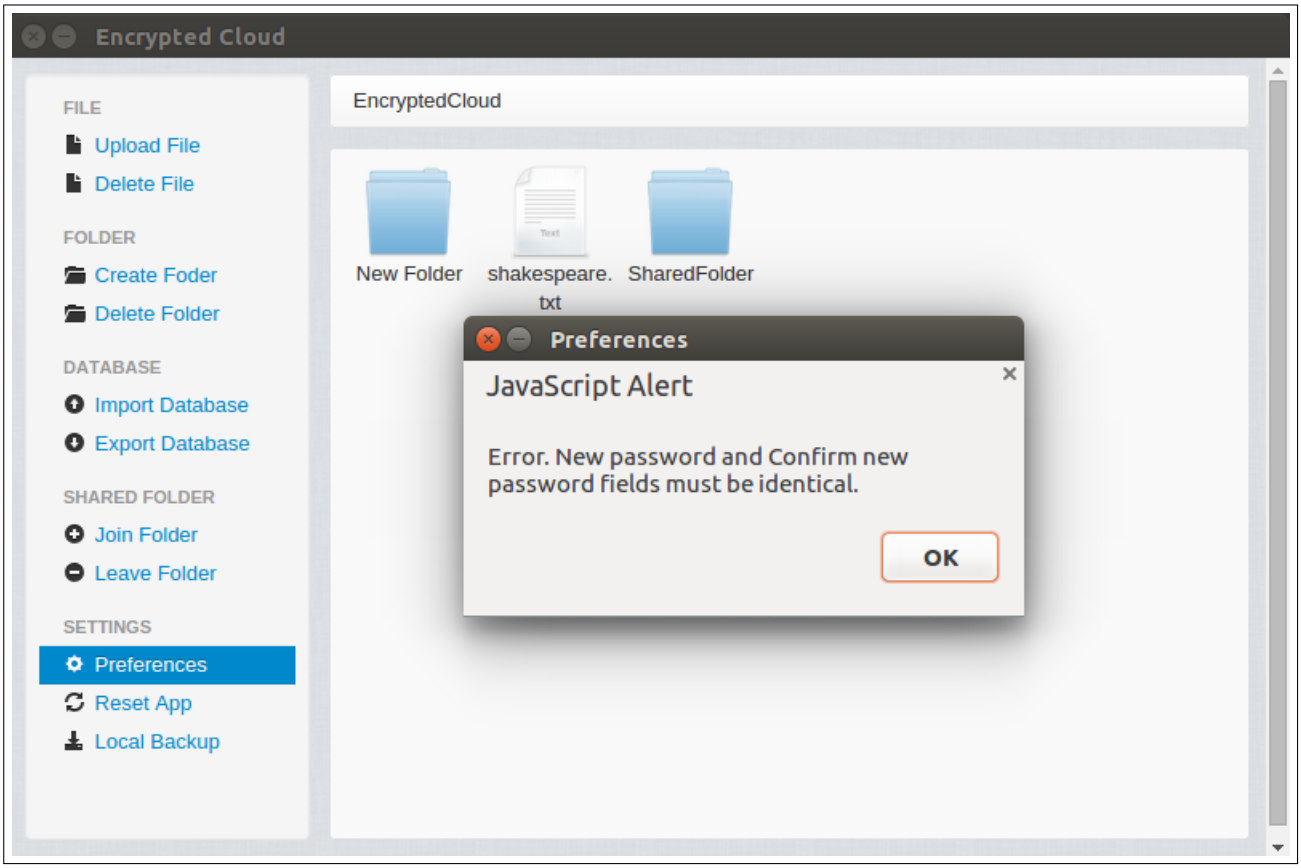


Figure C.29: Password modification integration test (VI).

Once the password is changed successfully (it is changed to "alejandro"), all non-shared files and the database are re-encrypted (because their key depends on the user's password). To verify this behaviour, the application is closed, the user logs in with the new password, and verifies that she can still access the file "shakespeare.txt":

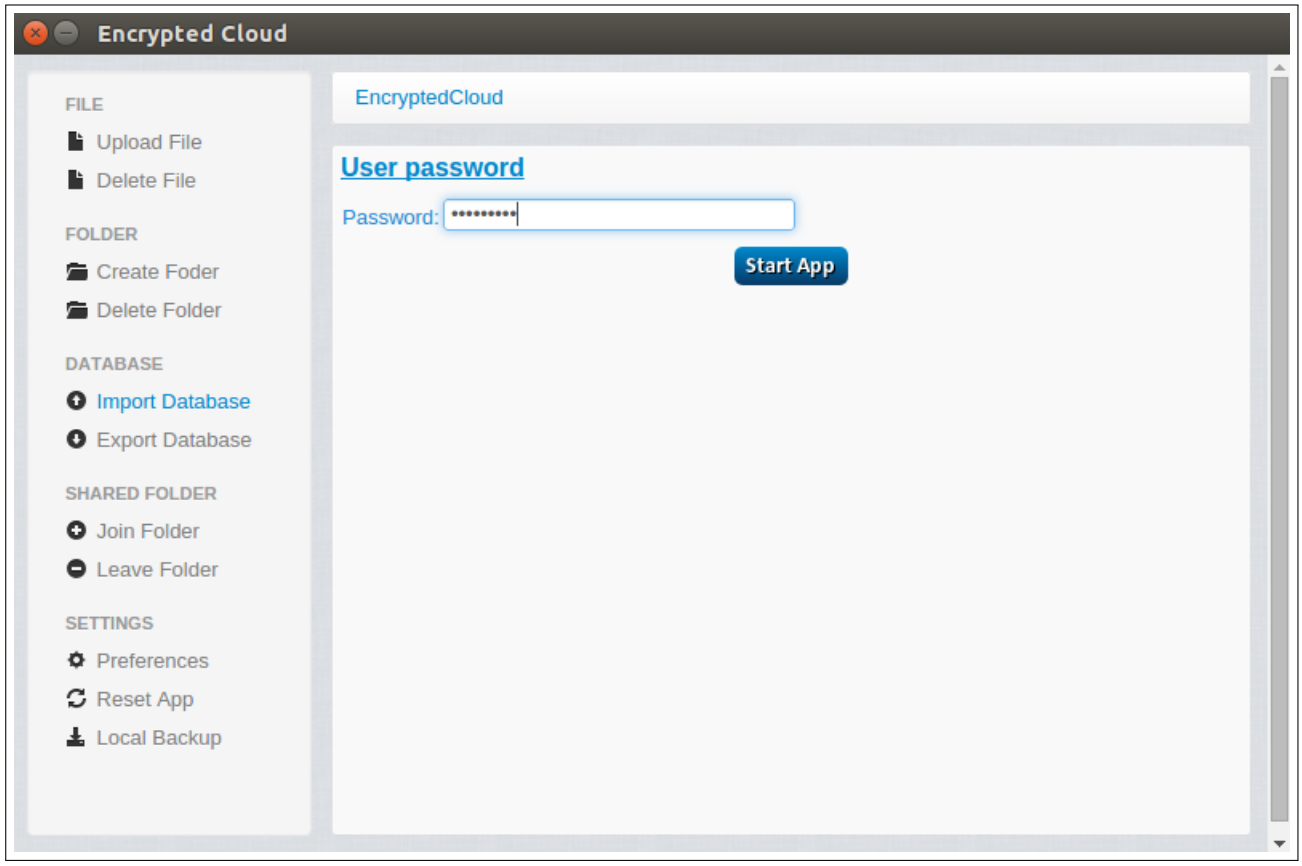


Figure C.30: Password modification integration test (VII).

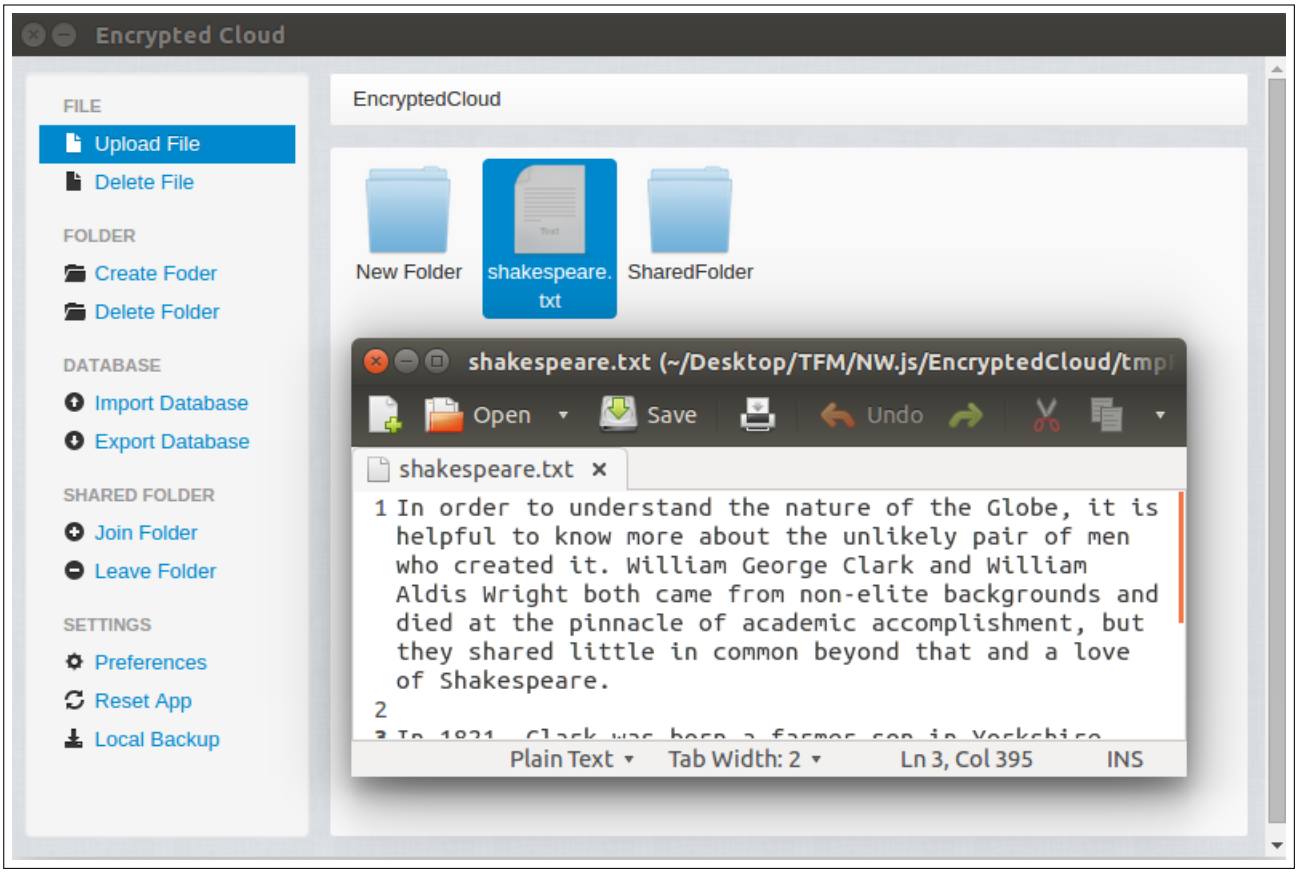


Figure C.31: Password modification integration test (VIII).

C.5 Unauthorised asset elimination integration test

It is important to ensure that when an asset is added to Encrypted Cloud it is not removed by an unauthorised third party. In this test, a file, called “tatras.jpg”, has been uploaded to the application root path.

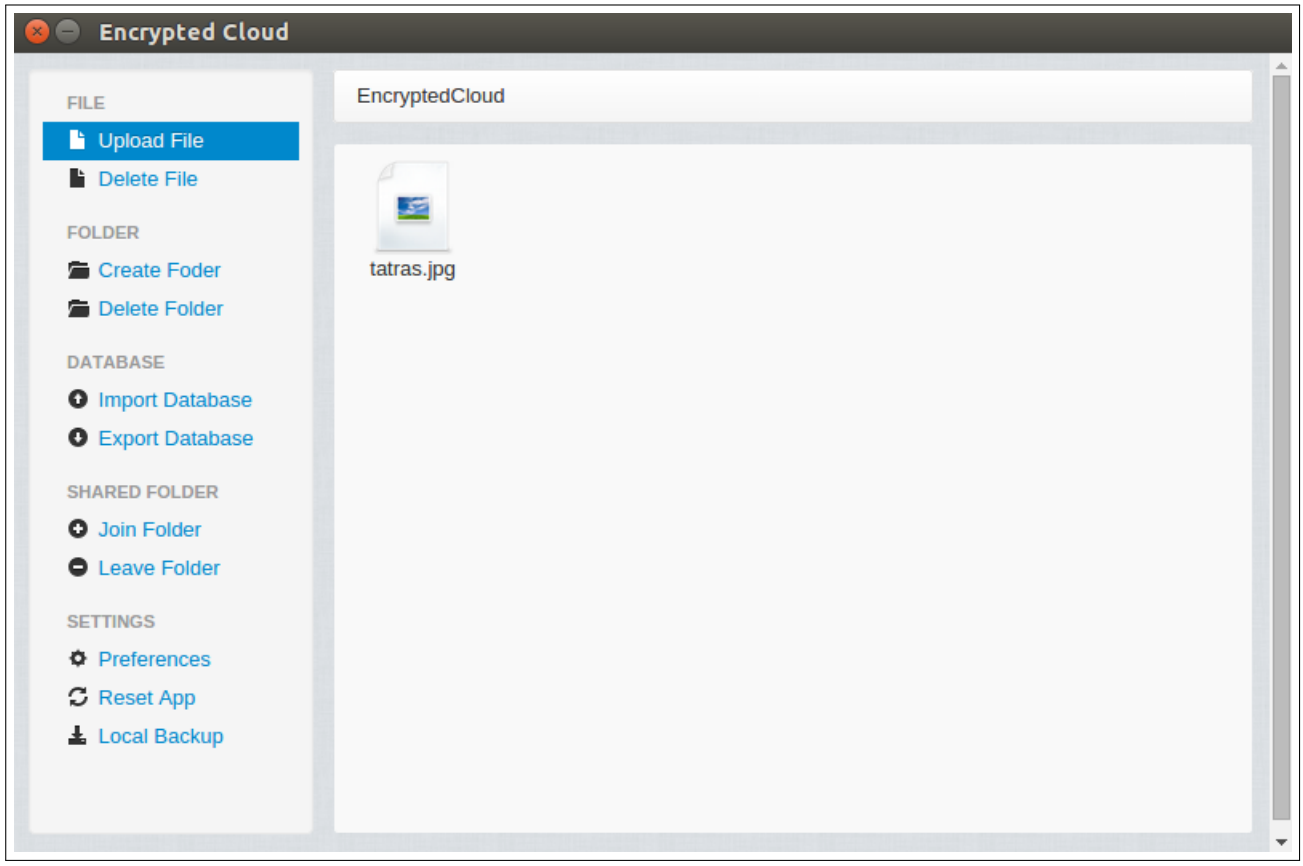


Figure C.32: Unauthorised asset elimination integration test (I).

This file is stored locally in the "Google Drive" folder, as shown in Figure C.33.

Figure C.33: Unauthorised asset elimination integration test (II).

To test this functionality, this file is manually deleted from its local directory:

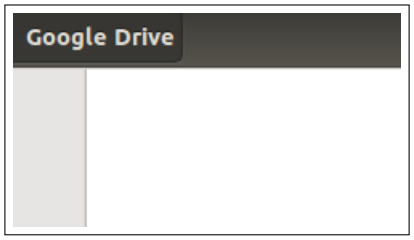


Figure C.34: Unauthorised asset elimination integration test (III).

Finally, it is checked that Encrypted Cloud displays a notification about this improper action:

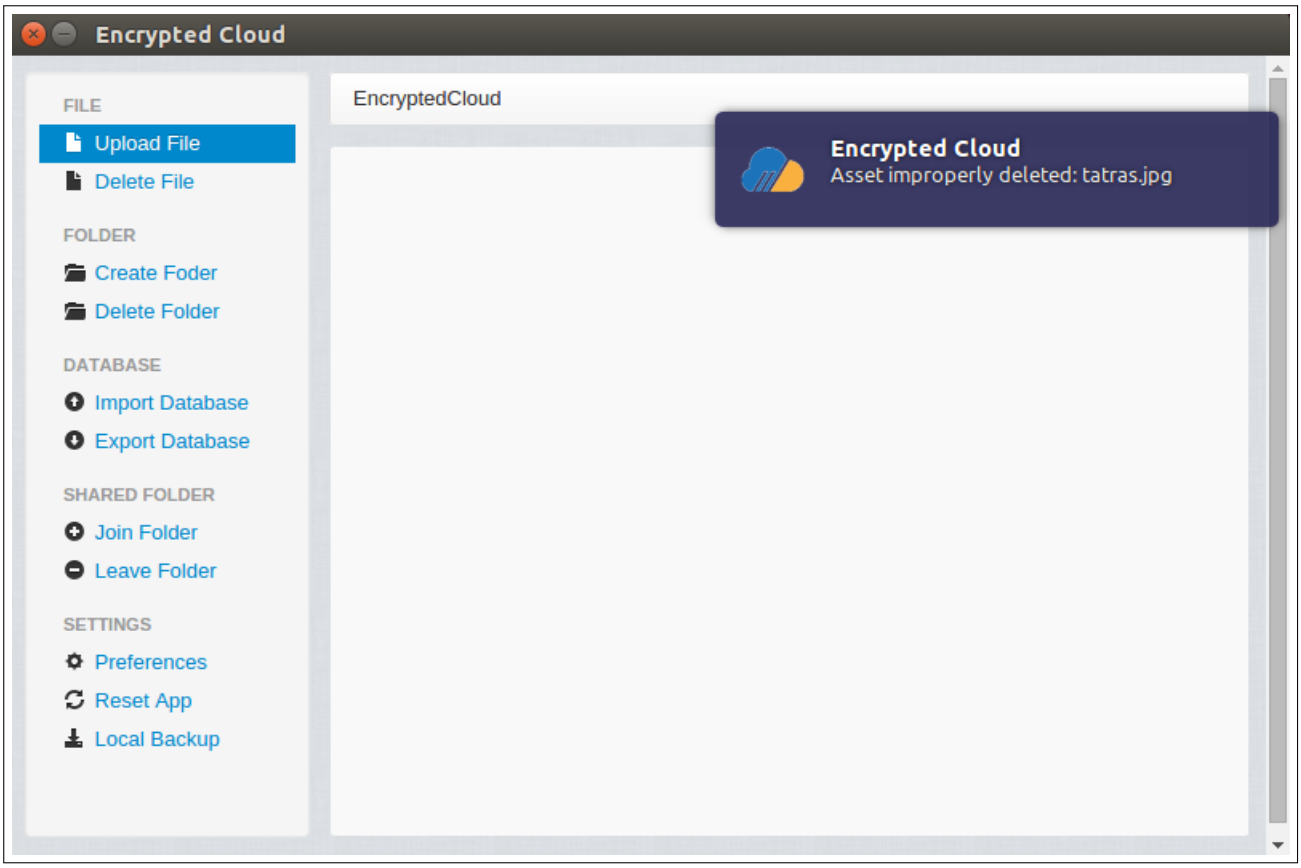


Figure C.35: Unauthorised asset elimination integration test (IV).

C.6 Encrypted Cloud reset integration test

The user can reset the application by clicking on the "Reset App" link on the left side panel. In this test, the user only has a file, "shakespeare.txt", which is stored locally in the non-shared directory "Google Drive" (Figure C.36 and Figure C.37).

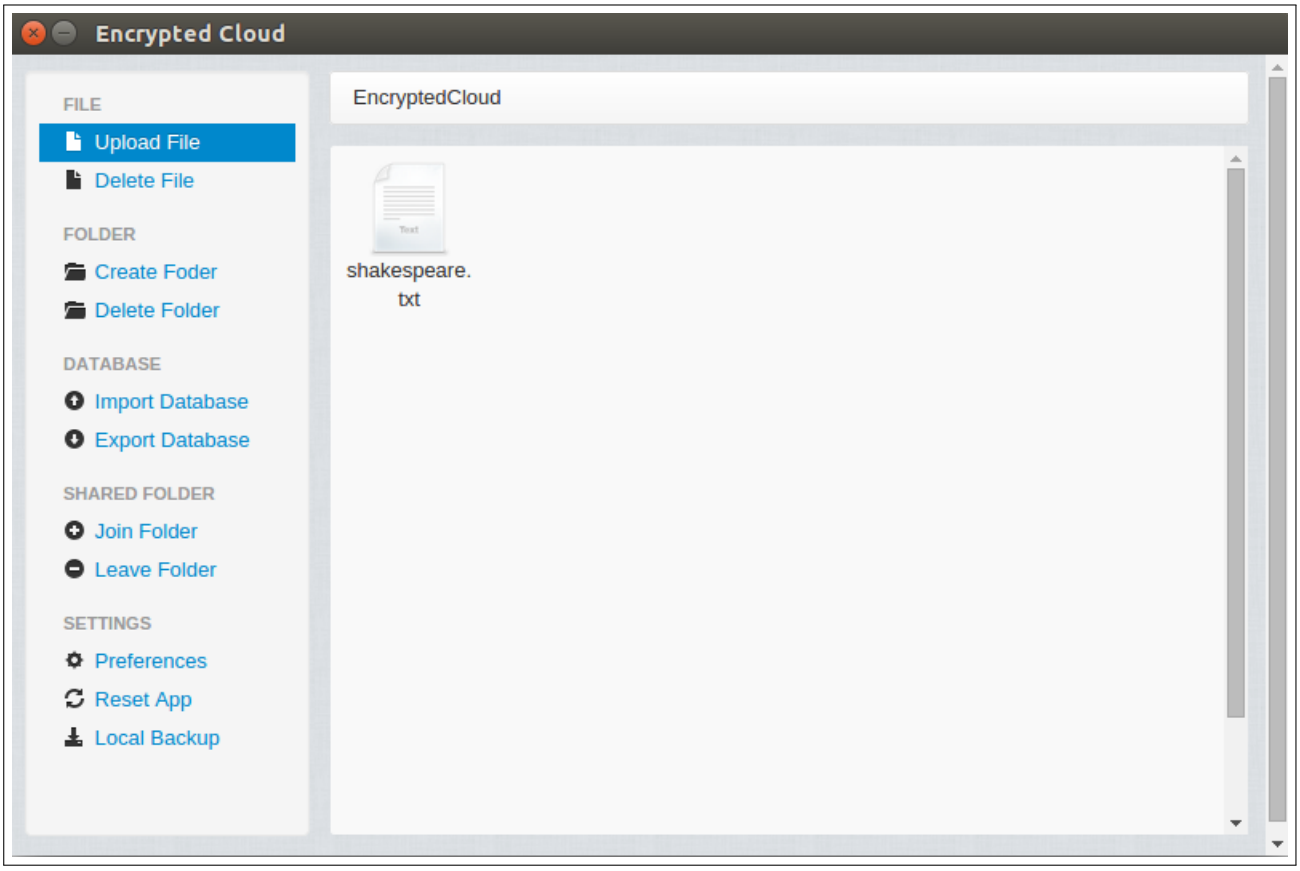


Figure C.36: Encrypted Cloud reset integration test (I).

Figure C.37: Encrypted Cloud reset integration test (II).

The user decides to reset the application, and after pressing on the "Reset App" link, it is displayed a pop-up with which the user would have the possibility to make a local backup before resetting the application. In this case, the user does not want to do this backup:

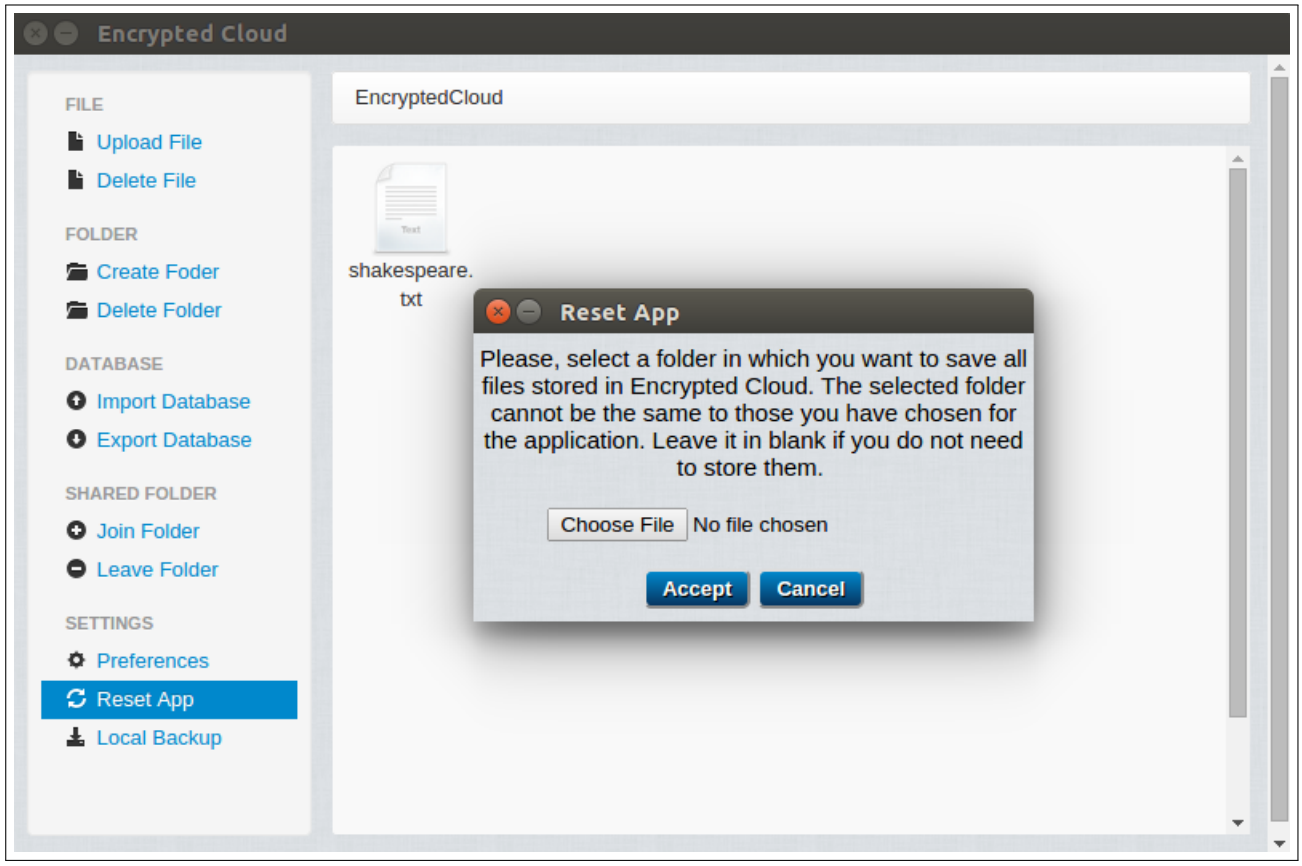


Figure C.38: Encrypted Cloud reset integration test (III).

After pressing the "Accept" button, the application is closed, and all those assets that the user has previously uploaded are deleted. In this case, it is verified that the file "shakespeare.txt" has been removed:

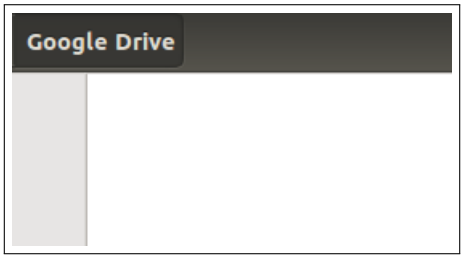


Figure C.39: Encrypted Cloud reset integration test (IV).

Moreover, since when the application is reset it is also removed the local database, if the user would start again Encrypted Cloud, it will be shown the initial configuration page:

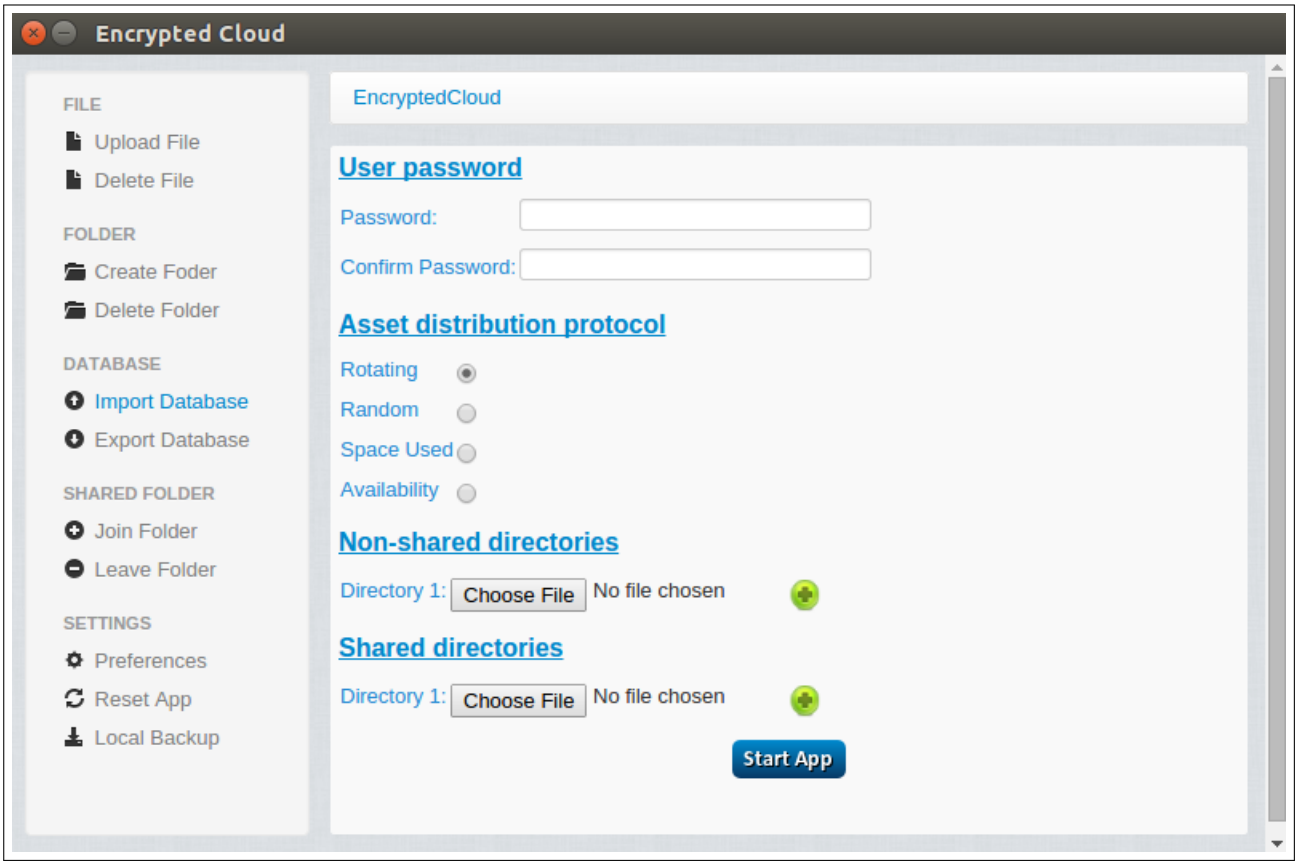


Figure C.40: Encrypted Cloud reset integration test (V).

C.7 Local backup integration test

To perform this test, the user chooses as non-shared folder “Dropbox” and as a shared folder “Dropbox/Shared Folder”. In addition, the user has two files at this moment, “tatras.jpg” located in the root path of the application, and “chain_bridge.jpg” stored in “Shared Folder”:

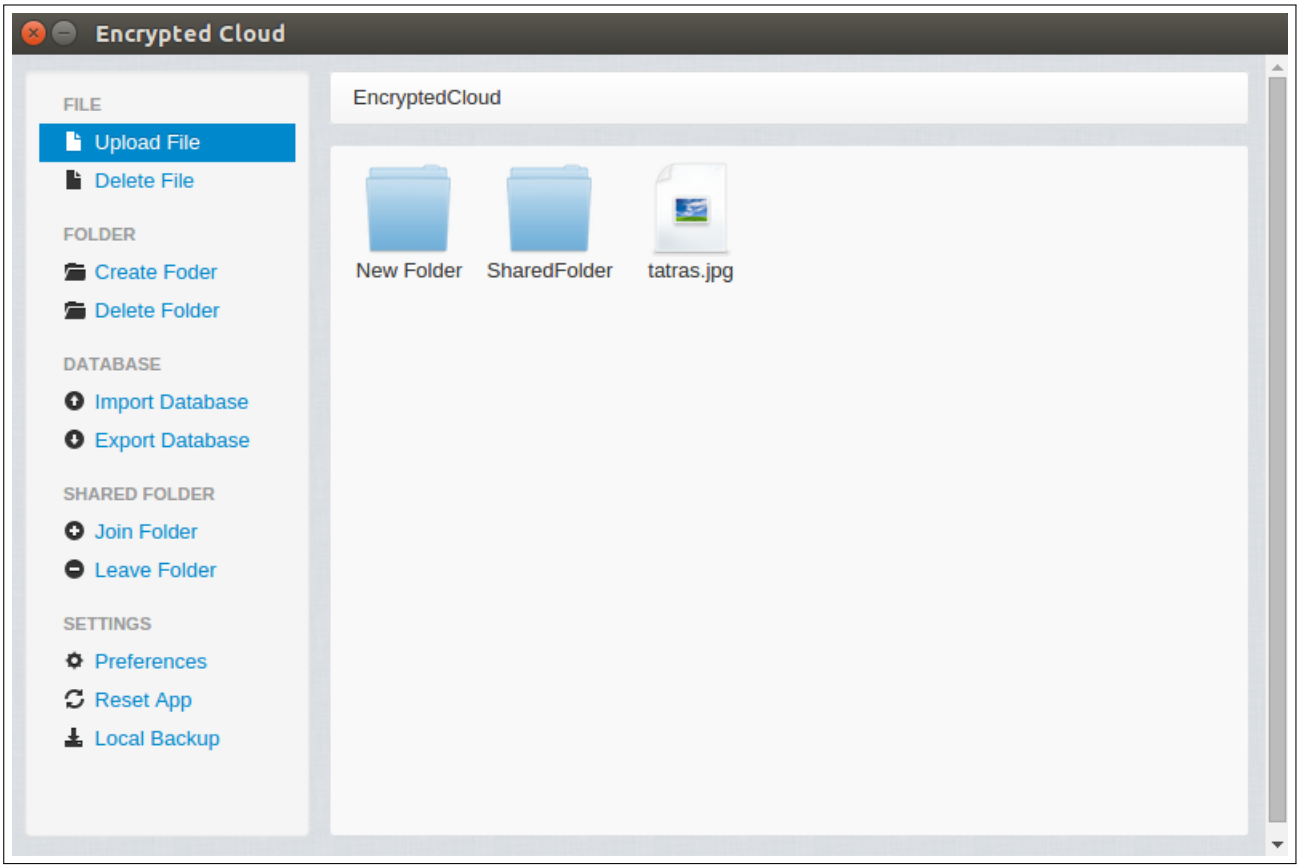


Figure C.41: Local backup integration test (I).

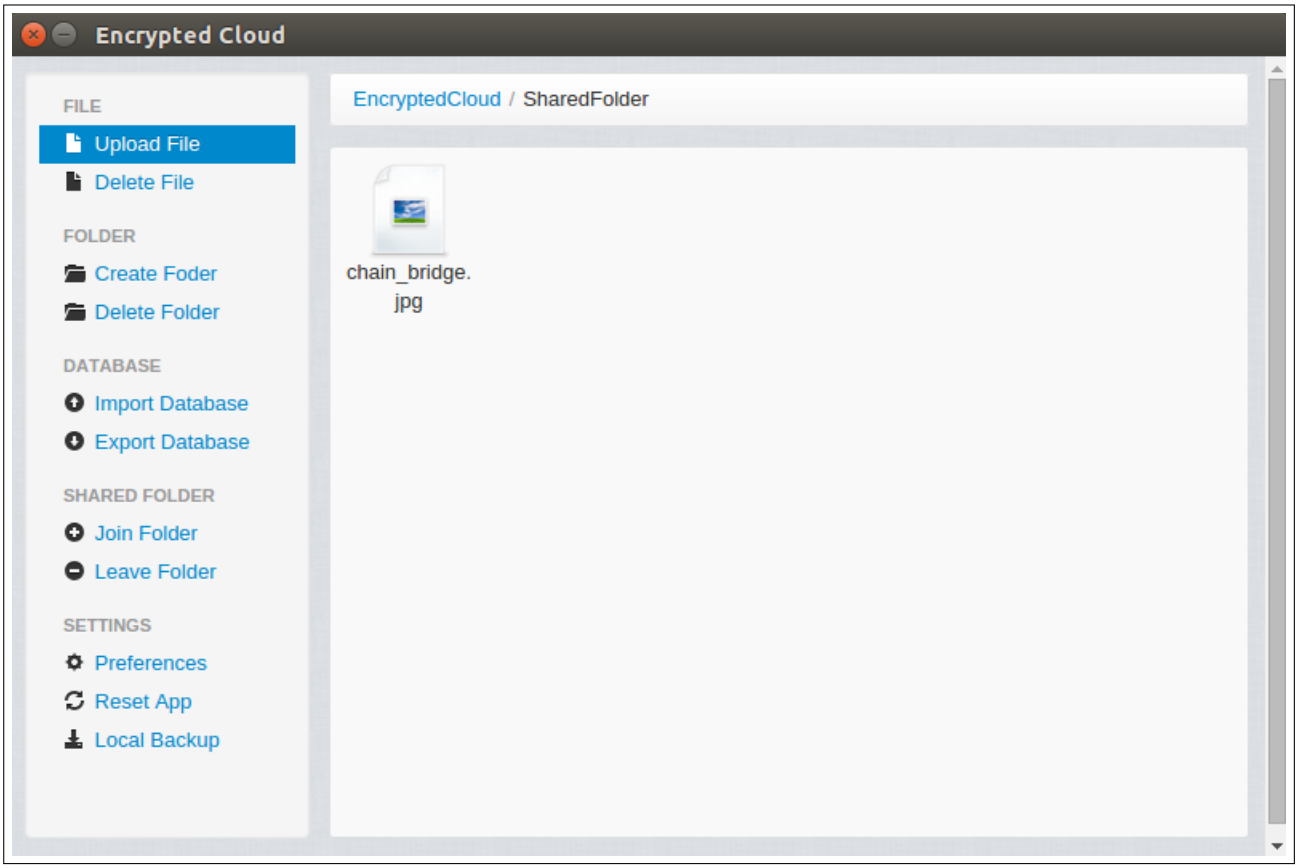


Figure C.42: Local backup integration test (II).

Then, the user decides to perform a local backup of the application by clicking on the “Local Backup” link which shows the following pop-up:

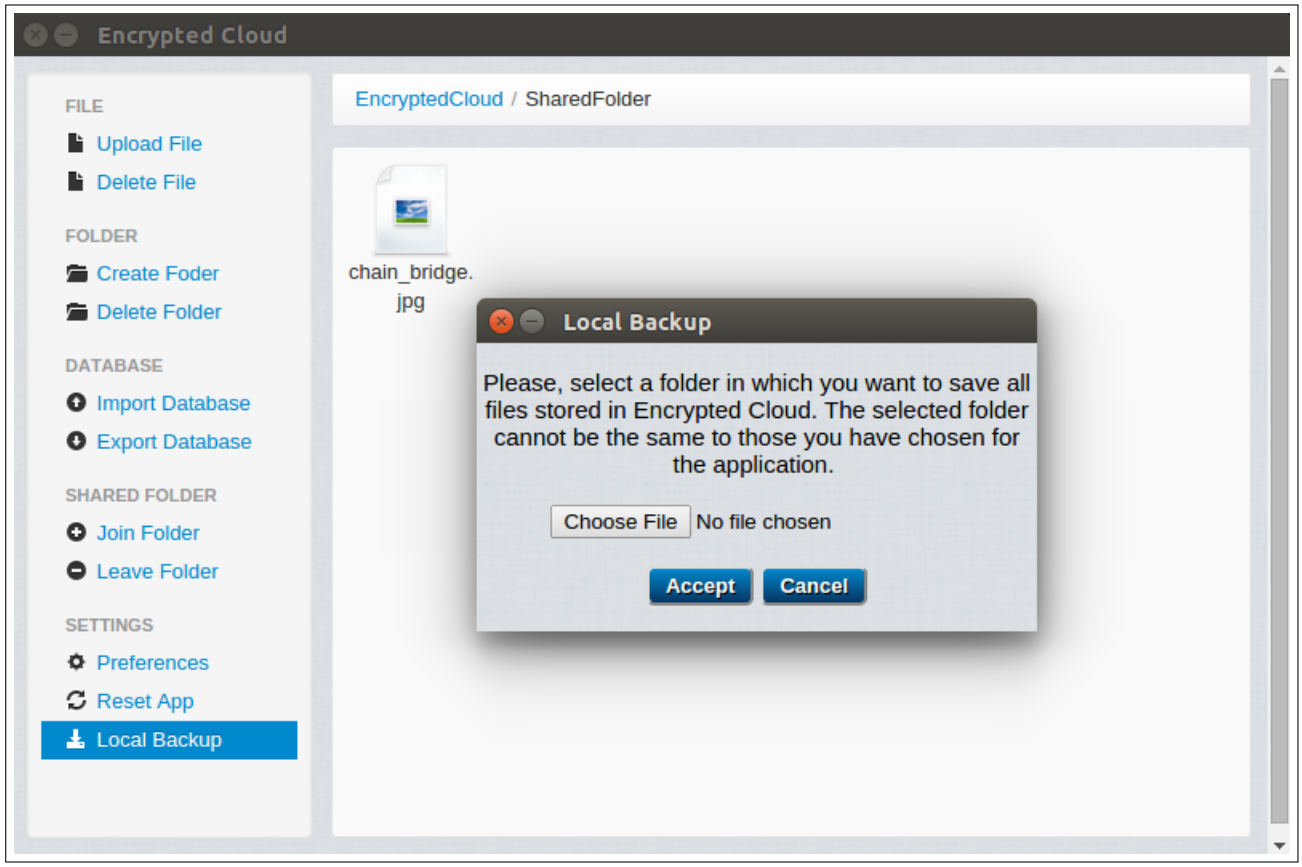


Figure C.43: Local backup integration test (III).

This pop-up verifies that the chosen directory is not empty (Figure C.44) and that it is not any of the shared or non-shared directories of the application (Figure C.45).

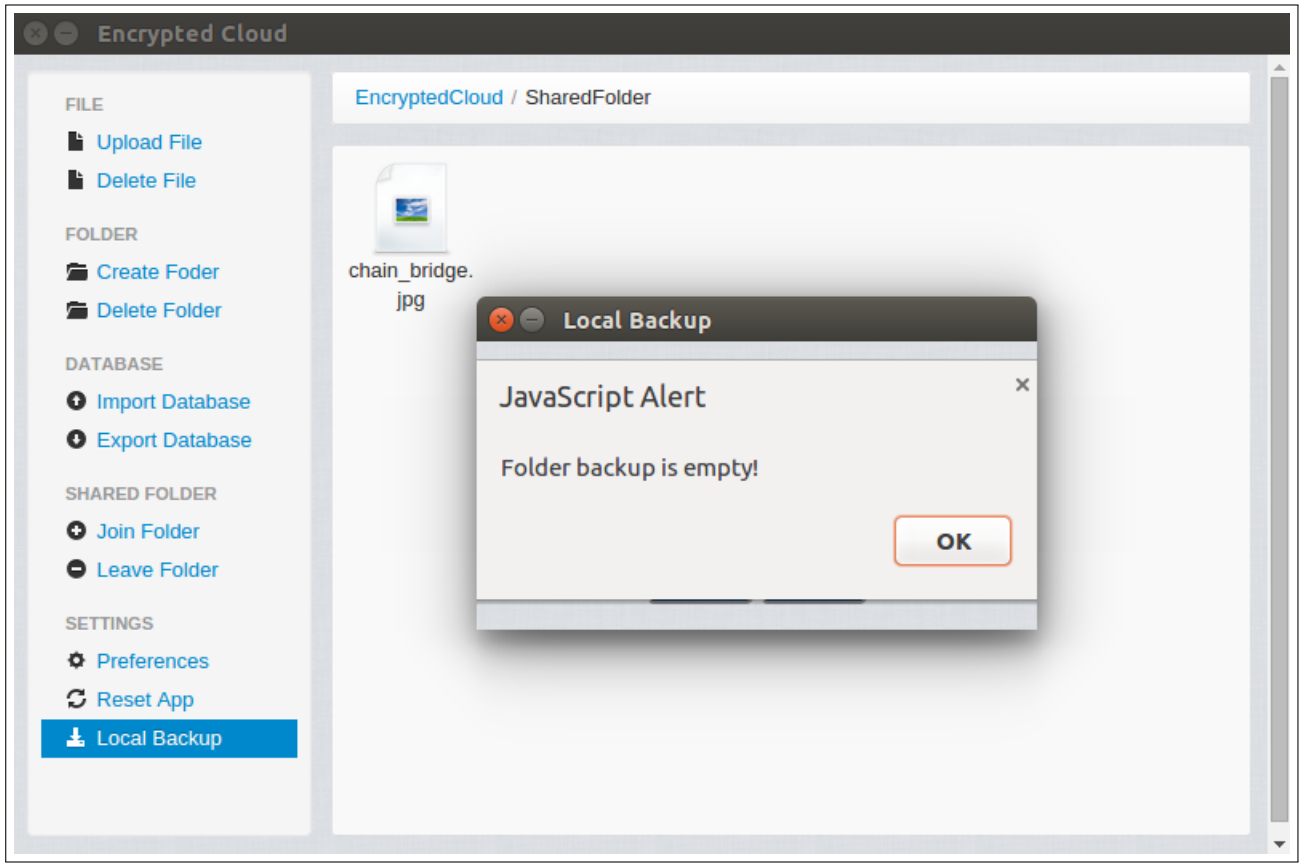


Figure C.44: Local backup integration test (IV).

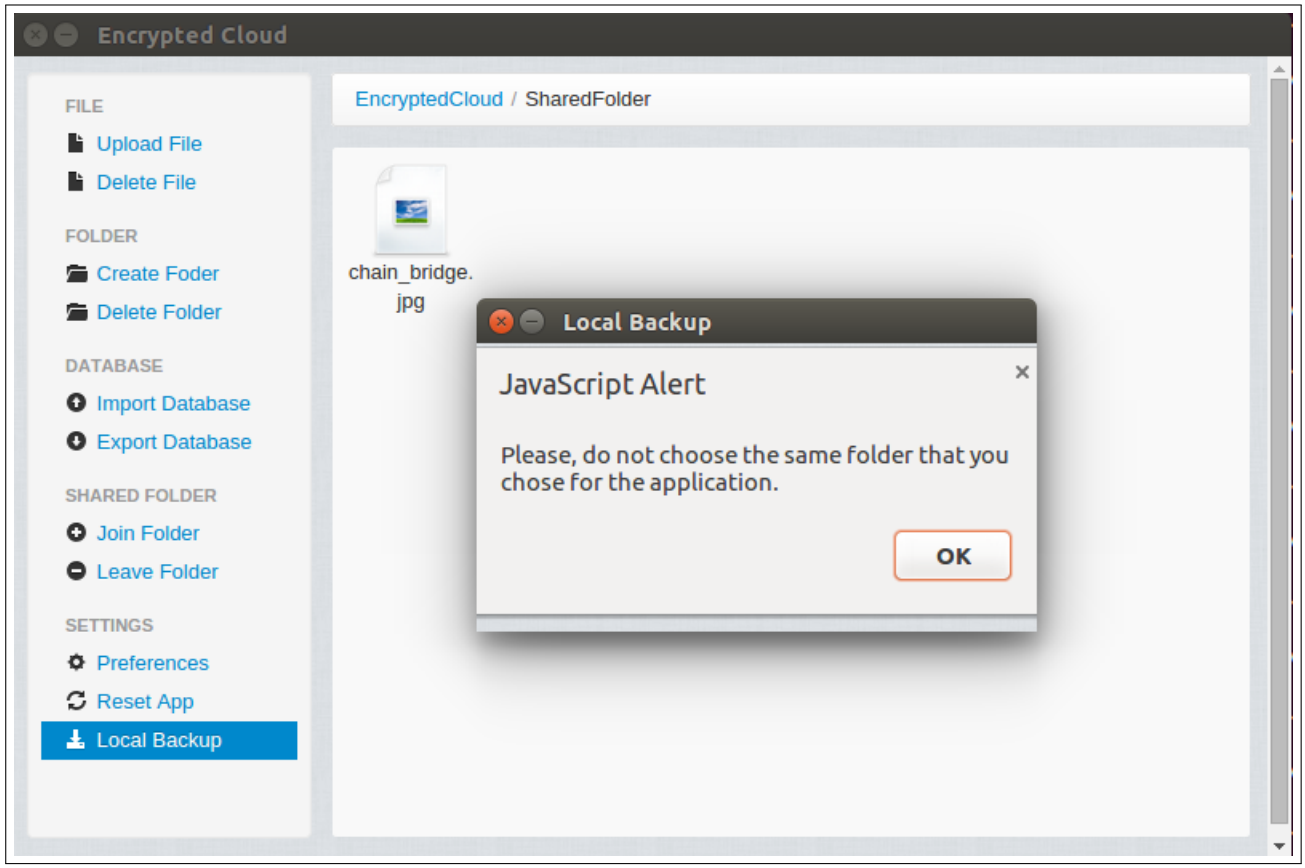


Figure C.45: Local backup integration test (V).

If finally the selected path is correct, the local backup is performed. When it is completed, it is notified to the user:

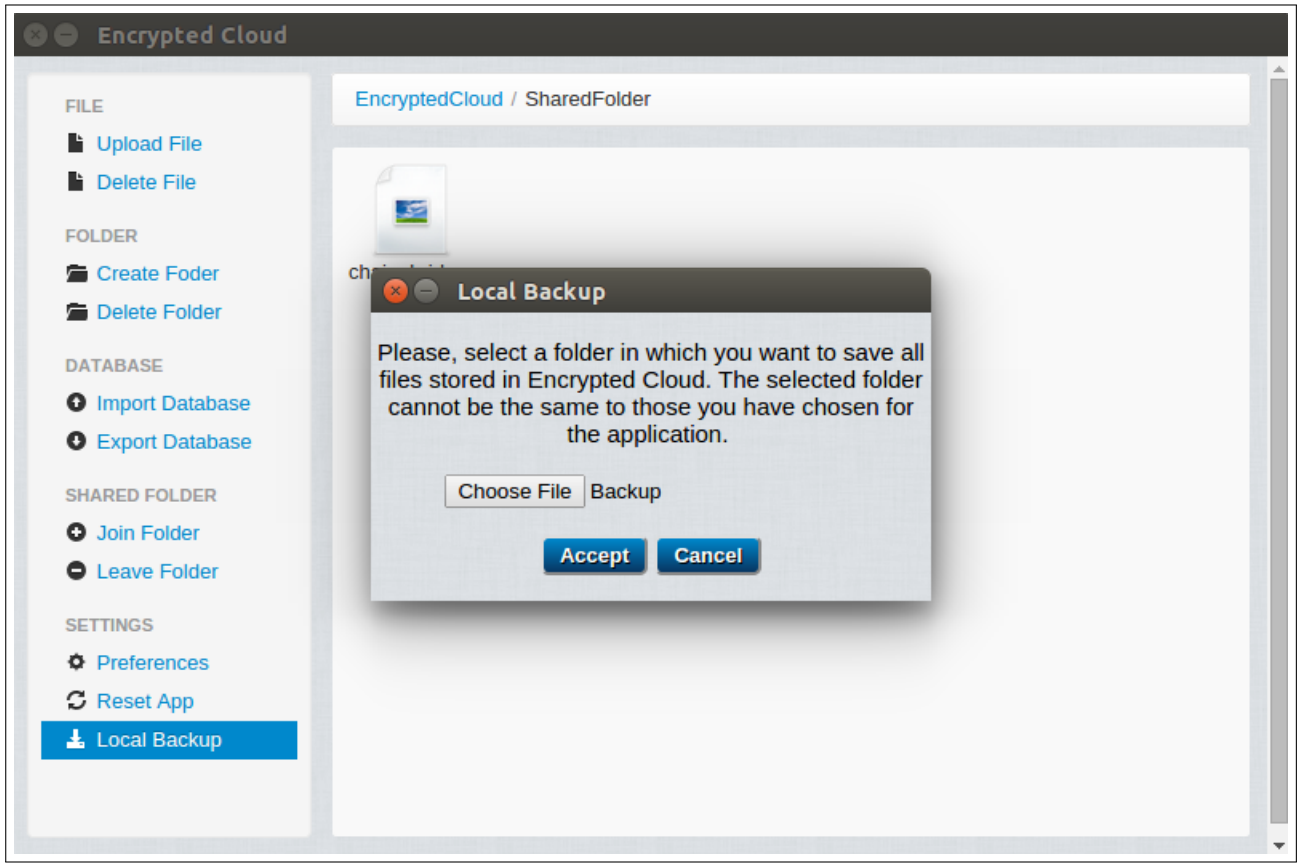


Figure C.46: Local backup integration test (VI).

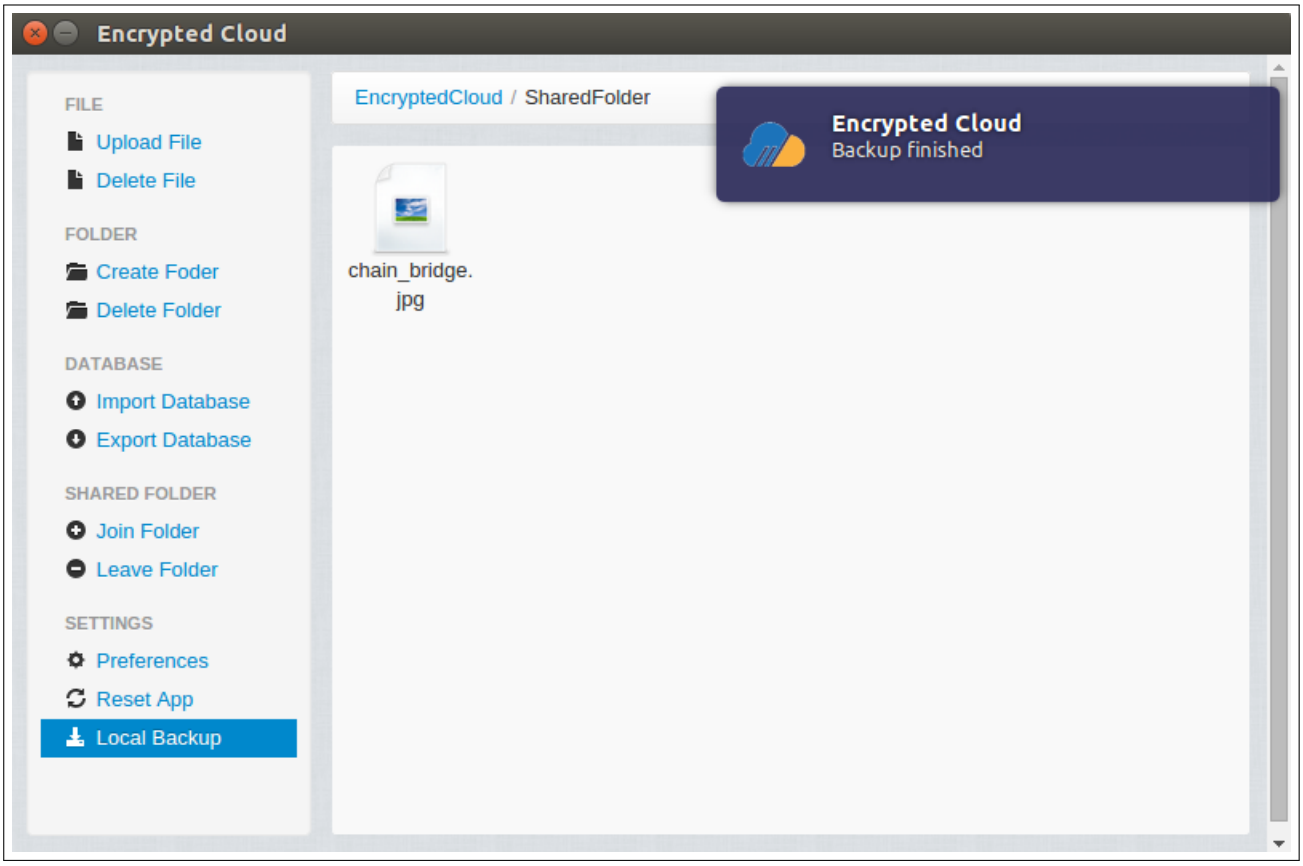


Figure C.47: Local backup integration test (VII).

Finally, it is verified that a local folder is created with all the files and folders that the user has in Encrypted Cloud:

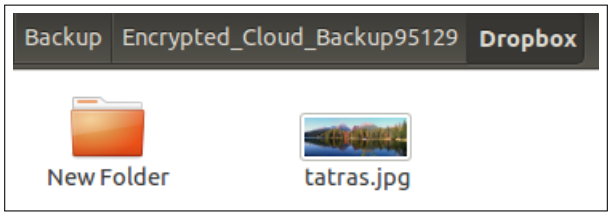


Figure C.48: Local backup integration test (VIII).

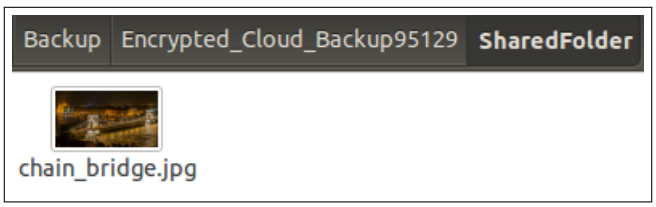


Figure C.49: Local backup integration test (IX).

C.8 Export database integration test

This test verifies that the user can export the current state of the local database to a folder named "DB backup". First, it is checked that this local folder is empty:

Figure C.50: Export database integration test (I).

Then, the user clicks on the link "Export Database" and selects "DB backup" as destination directory (Figure C.51).

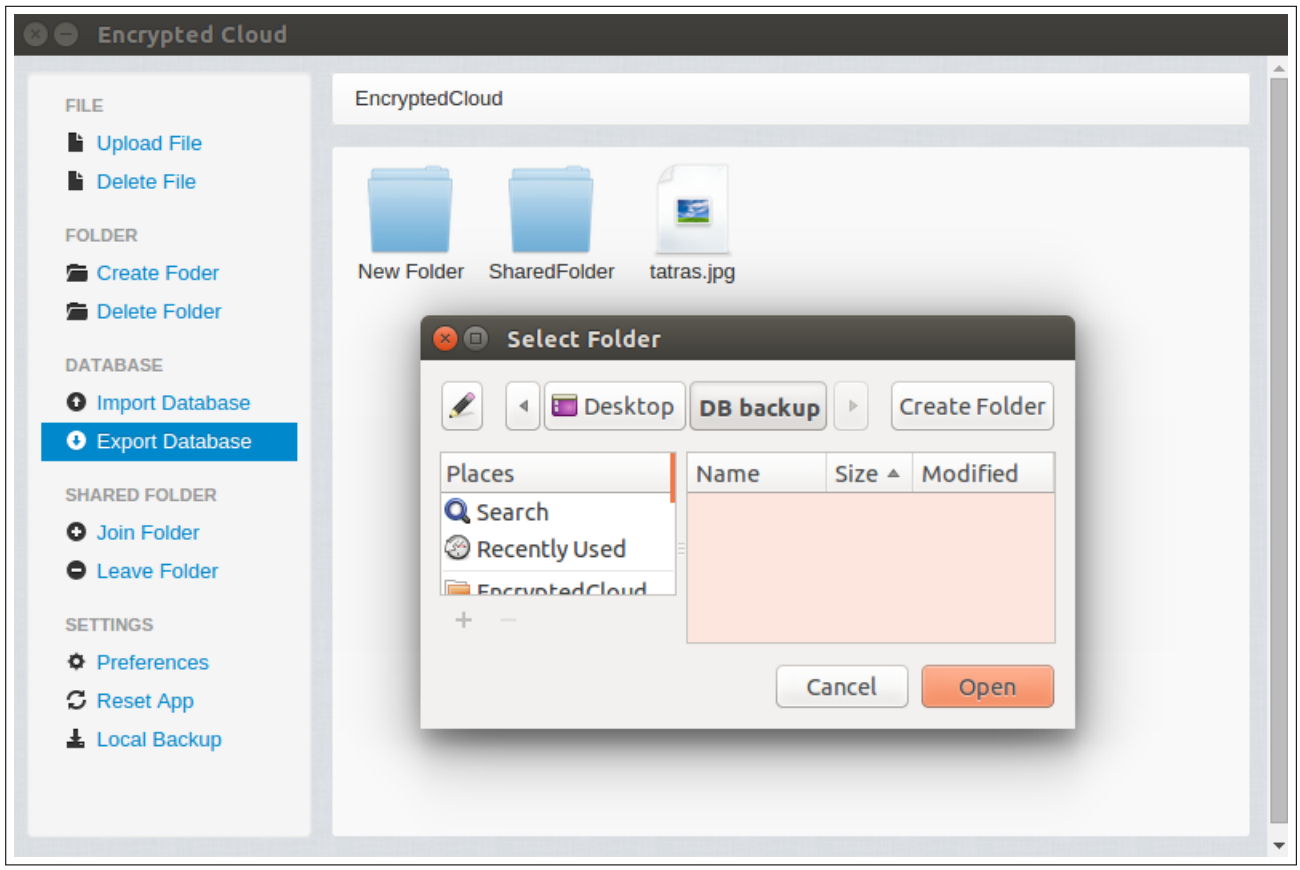


Figure C.51: Export database integration test (II).

Finally, the user clicks on the "Open" button and the database is exported:

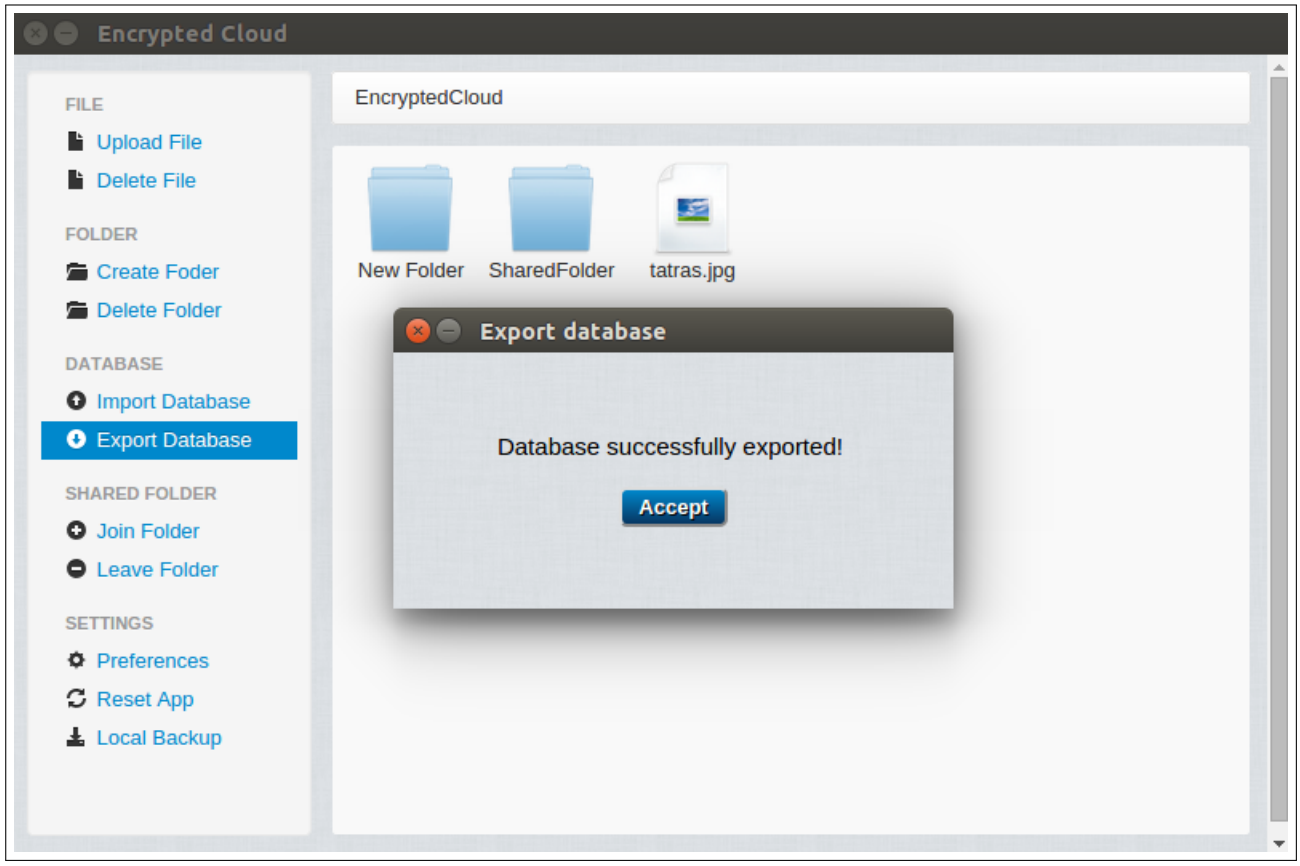


Figure C.52: Export database integration test (III).

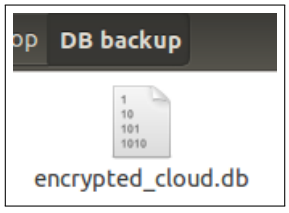


Figure C.53: Export database integration test (IV).

C.9 Import database integration test

In this test, the user has previously made a database export and then she has reset the application. However, she regrets about the last action. After this situation, when the user starts the application, it is loaded the initial configuration view of Encrypted Cloud:

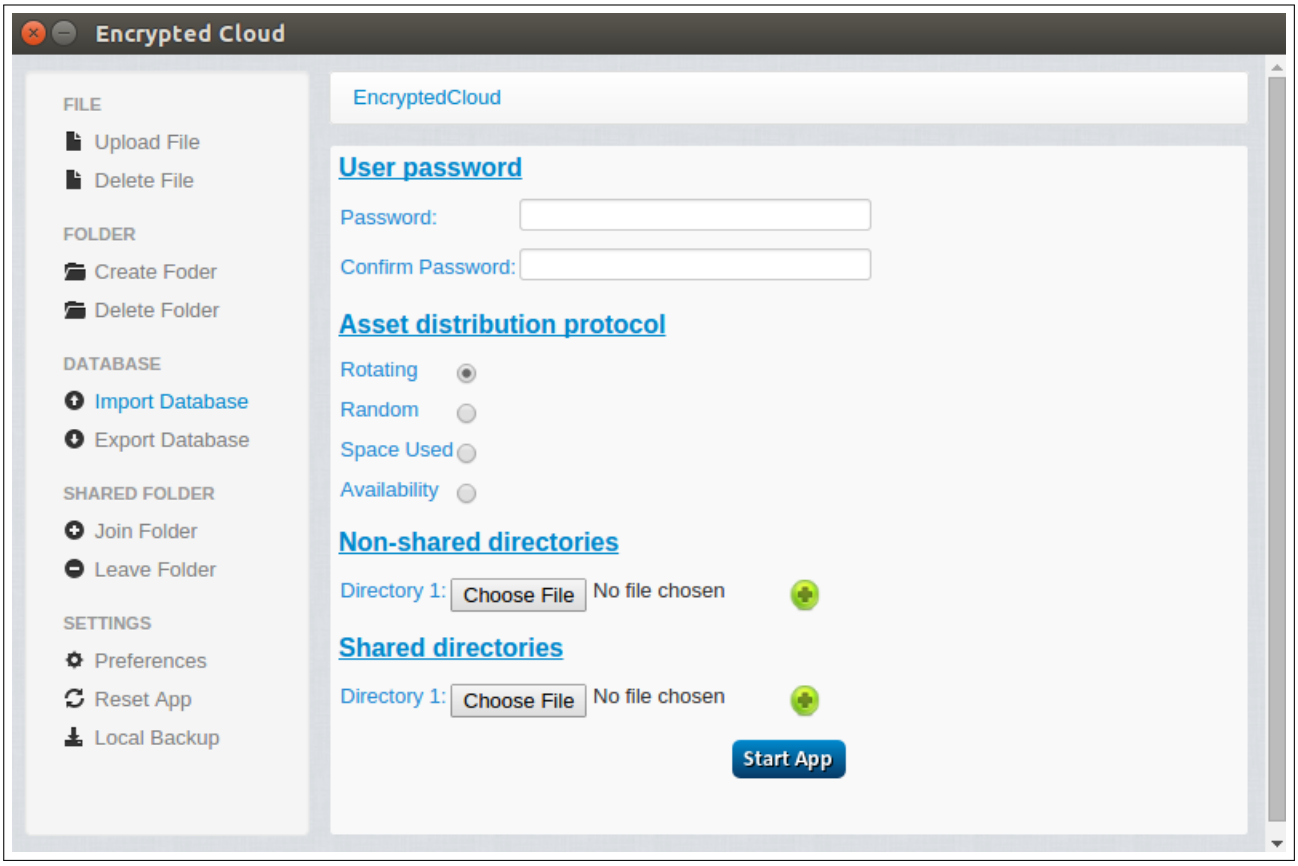


Figure C.54: Import database integration test (I).

Nevertheless, from this screen, the user could import a previous database. To do this, she has to click on the "Import Database" link and select the database that she wants to import and then press the "Open" button:

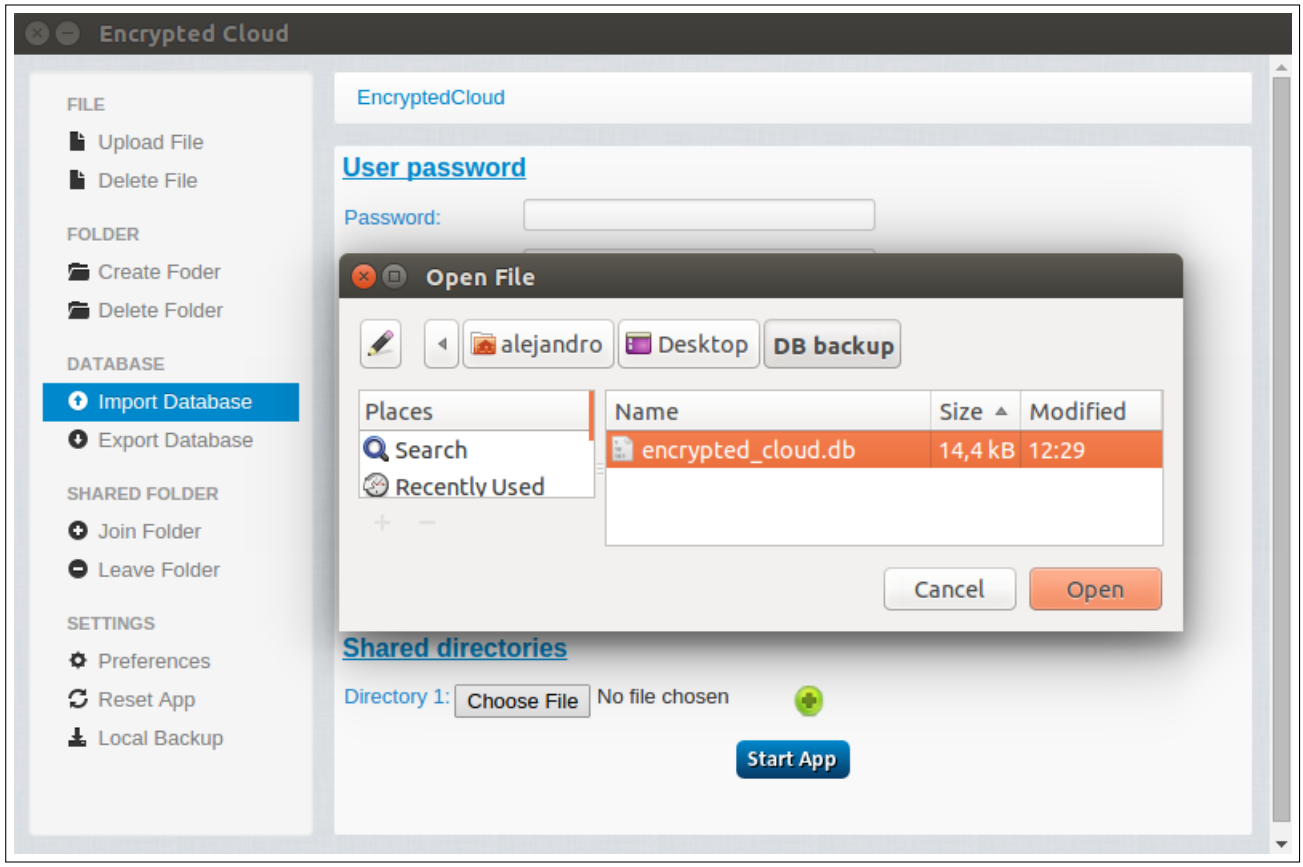


Figure C.55: Import database integration test (II).

After this action, the user can login with the password that she had when she exported this database backup:

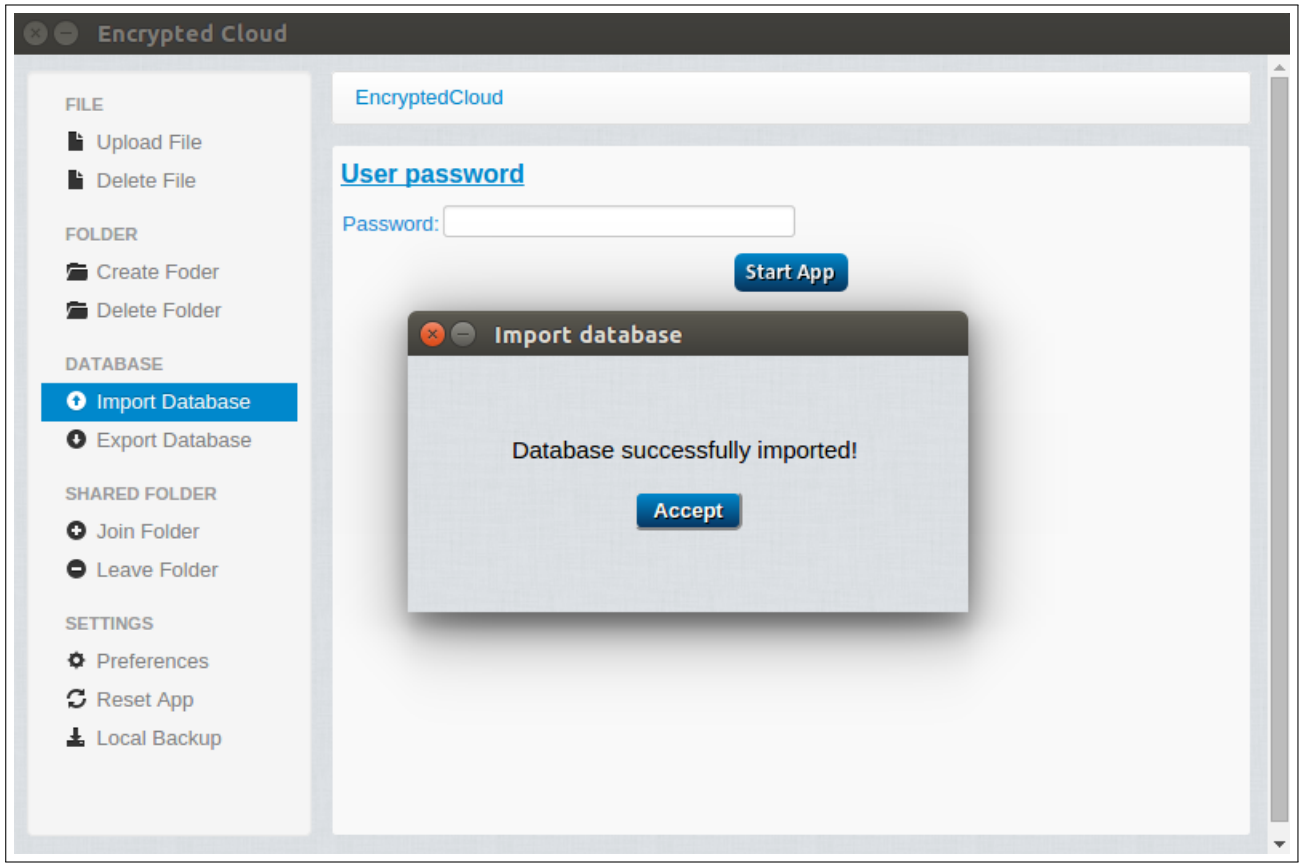


Figure C.56: Import database integration test (III).

Bibliography

[1] Ministerio de Economía y Competitividad de España, Plan estatal de investigación científica, técnica y de innovación 2013-2016, [Online]. Available from: http://www.idi.mineco.gob.es/stfls/MICINN/Investigacion/FICHEROS/Politicas_I+D+i/Plan_Estatal_Inves_cientifica_tecnica_innovacion.pdf [Accessed 2014-06-19]. (Cited on page 1)

[2] L. Siegele, S. Crosby, B. Schneier, Cloud computing — Debates, [Online]. Available from: http://debates.economist.com/debate/cloud-computing?state=rebuttal [Accessed 2016-04-20]. (Cited on page 1)

[3] InformationWeek, 9 Spectacular Cloud Computing Fails, [Online]. Available from: http://www.informationweek.com/cloud/9-spectacular-cloud-computing-fails/d/d-id/1321305 [Accessed 2016-05-25]. (Cited on page 1)

[4] Tech Worm, 700,000 Dropbox credentials hacked, [Online]. Available from: http://www.techworm.net/2014/10/700000-dropbox-credentials-hacked-hacker-leaks-dropbox-hacks-teasers-pastebin.html [Accessed 2014-10-22]. (Cited on page 2)

[5] European Commission, EU Cybersecurity plan to protect open internet and online freedom and opportunity - Cyber Security strategy and Proposal for a Directive, [Online]. Available from: http://ec.europa.eu/digital-agenda/en/news/eu-cybersecurity-plan-protect-open-internet-and-online-freedom-and-opportunity-cyber-security [Accessed 2014-06-15]. (Cited on page 2)

[6] A. Sanchez-Gomez, Estudio del problema de la distribución de claves criptográficas en el contexto de las tecnologías cloud, [Online]. Available from: https://repositorio.uam.es/xmlui/handle/10486/662513 [Accessed 2014-09-19] (jul). (Cited on page 2, 14, 16)

[7] P. Regan, Self-Hosted Cloud-Storage Comparison - 2014 Edition - Patshead.com Blog, [Online]. Available from: http://blog.patshead.com/2014/09/self-hosted-cloud-storage-comparison-2014-edition.html [Accessed 2016-04-20] (Accessed 2016-04-20). (Cited on page 2)

[8] J.-H. Hoepman, B. Jacobs, Increased security through open source, Communications of the ACM 50 (1) (2007) 79–83. (Cited on page 2)

[9] Maketecheasier, MEGA: The Latest Cloud Storage Service That Beats The Rest, [Online]. Available from: https://www.maketecheasier.com/mega-cloud-storage-review/ [Accessed 2016-02-05]. (Cited on page 3)

[10] MEGApwn, MEGApwn, [Online]. Available from: https://nzkoz.github.io/MegaPWN/ [Accessed 2014-06-19]. (Cited on page 3)

[11] S. Tamrakar, L. Nguyen, P. K. Pendyala, A. Paverd, N. Asokan, A.-R. Sadeghi, OmniShare: Securely Accessing Encrypted Cloud Storage from Multiple Authorized Devices (2015) 10.
URL http://arxiv.org/abs/1511.02119 (Cited on page 3)

[12] K. Goda, M. Kitsuregawa, The History of Storage Systems., Proceedings of the IEEE 100 (Centennial-Issue) (2012) 1433–1440. (Cited on page 5)

[13] J. Gleick, The Information: A History, a Theory, a Flood, Pantheon Books, 2011. (Cited on page 5)

[14] S. Foresti, Preserving privacy in data outsourcing, Vol. 99, Springer Science & Business Media, 2010. (Cited on page 5)

[15] C. Diaz, O. Tene, S. Gurses, Hero or villain: The data controller in privacy law and technologies, Ohio St. LJ 74 (2013) 923. (Cited on page 5)

[16] R. C. Martin, Clean code: a handbook of agile software craftsmanship, Pearson Education, 2009. (Cited on page 5)

[17] S. Islam, M. Ouedraogo, C. Kalloniatis, H. Mouratidis, S. Gritzalis, Assurance of Security and Privacy Requirements for Cloud Deployment Model. (Cited on page 6)

[18] A. Shostack, Threat modeling: Designing for security, John Wiley & Sons, 2014. (Cited on page 6)

[19] P. Mell, T. Grance, The NIST definition of cloud computing, Special Publication 800-145, NIST. (Cited on page 6)

[20] W. Jansen, T. Grance, Others, Guidelines on security and privacy in public cloud computing, NIST special publication 800 (2011) 144. (Cited on page 6)

[21] A. Ziv, The Ethereal Perimeter, [Online]. Available from: https://www.linkedin.com/pulse/ethereal-perimeter-avishai-ziv [Accessed 2015-11-22]. (Cited on page 7, 15)

[22] Securosis, Pragmatic Security for Cloud and Hybrid Networks, [Online]. Available from: https://securosis.com/blog/pragmatic-security-for-cloud-and-hybrid-networks-introduction [Accessed 2016-02-07]. (Cited on page 7)

[23] International Business Times, As Hackers Increasingly Target The Cloud, Rackspace Turns To Military Vet With Cyberwar Experience, [Online]. Available from: http://www.ibtimes.com/hackers-increasingly-target-cloud-rackspace-turns-military-vet-cyberwar-experience-2139057 [Accessed 2016-05-19]. (Cited on page 7)

[24] Eurostat, Cloud computing - statistics on the use by enterprises, [Online]. Available from: http://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises [Accessed 2016-05-19]. (Cited on page 8)

[25] Hold Security, Hold Security Recovers 272 Million Stolen Credentials From A Collector, [Online]. Available from: http://holdsecurity.com/news/the_collector_breach/ [Accessed 2016-05-17]. (Cited on page 8)

[26] Fortune, LinkedIn Data Breach: 117 Million Emails and Passwords Leaked, [Online]. Available from: http://fortune.com/2016/05/18/linkedin-data-breach-email-password/ [Accessed 2016-05-19]. (Cited on page 8)

[27] Imperva, Man in the Cloud (MITC) Attacks 1–16.
URL https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf (Cited on page 8, 10)

[28] W. Li, C. J. Mitchell, Security Issues in OAuth 2.0 SSO Implementations, Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings (2014) 529–541. (Cited on page 9, 10)

[29] C. Ruvalcaba, C. Langin, Four Attacks on OAuth - How to Secure Your OAuth Implementation, System (1) (2009) 19.
URL https://www.sans.org/reading-room/whitepapers/application/attacks-oauth-secure-oauth-implementation-33644 (Cited on page 9, 10)

[30] The Guardian, Naked celebrity hack: security experts focus on iCloud backup theory, [Online]. Available from: http://www.theguardian.com/technology/2014/sep/01/naked-celebrity-hack-icloud-backup-jennifer-lawrence [Accessed 2016-01-08]. (Cited on page 9)

[31] M. Abdalla, P.-A. Fouque, D. Pointcheval, Password-based authenticated key exchange in the three-party setting, in: Public Key Cryptography-PKC 2005, Springer, 2005, pp. 65–84. (Cited on page 9, 168)

[32] Stanford University, SRP: What Is It?, [Online]. Available from: http://srp.stanford.edu/whatisit.html [Accessed 2015-09-19]. (Cited on page 9, 170)

[33] W. GORDON, Two-Factor Authentication: The Big List Of Everywhere You Should Enable It Right Now, [Online]. Available from: http://www.lifehacker.com.au/2012/09/two-factor-authentication-the-big-list-of-everywhere-you-should-enable-it-right-now/ [Accessed 2015-12-31]. (Cited on page 9)

[34] B. info, Google Authenticator What is google authenticator?, [Online]. Available from: https://blockchain.info/nl/wallet/google-authenticator [Accessed 2015-11-15]. (Cited on page 9)

[35] Alphr, How secure are Dropbox, Microsoft OneDrive, Google Drive and Apple iCloud?, [Online]. Available from: http://www.alphr.com/dropbox/1000326/how-secure-are-dropbox-microsoft-onedrive-google-drive-and-apple-icloud [Accessed 2015-12-29]. (Cited on page 10)

[36] Box, Can I Enable 2-Step Verification For My Box Account?, [Online]. Available from: https://community.box.com/t5/Account-Information/Can-I-Enable-2-Step-Verification-For-My-Box-Account/ta-p/29 [Accessed 2016-01-02]. (Cited on page 10)

[37] T. Pulls, D. Slamanig, On the Feasibility of (Practical) Commercial Anonymous Cloud Storage, Transactions on Data Privacy 8 (2) (2015) 89–111. (Cited on page 10)

[38] D. Arroyo, J. Diaz, F. B. Rodriguez, Non-conventional Digital Signatures and Their Implementations–A Review (2013) 2013. (Cited on page 10, 16, 17)

[39] KuppingerCole, Why we need Dynamic Authorization Management, [Online]. Available from: https://www.kuppingercole.com/blog/kuppinger/why-we-need-dynamic-authorization-management [Accessed 2016-05-18]. (Cited on page 10, 166)

[40] D. Fett, R. Küsters, G. Schmitz, A Comprehensive Formal Security Analysis of OAuth 2.0 (2016) 1–75.
URL http://arxiv.org/abs/1601.01229 (Cited on page 10)

[41] Netcraft, Half a million widely trusted websites vulnerable to Heartbleed bug, [Online]. Available from: http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html [Accessed 2015-11-22]. (Cited on page 10)

[42] Qualys Blog, The GHOST Vulnerability, [Online]. Available from: https://blog.qualys.com/laws-of-vulnerabilities/2015/01/27/the-ghost-vulnerability [Accessed 2016-02-06]. (Cited on page 11)

[43] A. Dmitrienko, C. Liebchen, C. Rossow, A.-R. Sadeghi, Security analysis of mobile two-factor authentication schemes, Intel® Technology Journal 18 (4). (Cited on page 11)

[44] Makeuseof, Two-Factor Authentication Hacked: Why You Shouldn't Panic, [Online]. Available from: http://www.makeuseof.com/tag/two-factor-authentication-hacked-shouldnt-panic/ [Accessed 2016-02-06]. (Cited on page 11)

[45] BankInfoSecurity, New Trojan Exploits Mobile Channel, [Online]. Available from: http://www.bankinfosecurity.com/interviews/new-trojan-exploits-mobile-channel-i-1730 [Accessed 2016-02-06]. (Cited on page 11)

[46] E. Kalige, D. Burkey, A Case Study of Eurograbber: How 36 Million Euros was Stolen via Malware, [Online]. Available from: https://www.checkpoint.com/download/downloads/products/whitepapers/Eurograbber_White_Paper.pdf [Accessed 2016-02-06]. (Cited on page 11)

[47] C. Mainka, V. Mladenov, F. Feldmann, J. Krautwald, J. Schwenk, Your software at my service: Security analysis of saas single sign-on solutions in the cloud, in: Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, ACM, 2014, pp. 93–104. (Cited on page 11)

[48] C. Mainka, V. Mladenov, T. Guenther, J. Schwenk, Automatic Recognition, Processing and Attacking of Single Sign-On Protocols with Burp Suite. (Cited on page 11)

[49] K. Strandburg, Monitoring, Datafication and Consent: Legal Approaches to Privacy in a Big Data Context, Privacy, Big Data, and the Public Good: Frameworks for Engagement (2014) 5. (Cited on page 11)

[50] P. Samarati, S. di Vimercati, Cloud security: Issues and concerns, Encyclopedia on Cloud Computing. Wiley, New York. (Cited on page 11, 12, 13)

[51] G. Yang, J. Yu, W. Shen, Q. Su, Z. Fu, R. Hao, Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability, Journal of Systems and Software 113 (2016) 130–139.
URL http://linkinghub.elsevier.com/retrieve/pii/S016412121500268X
(Cited on page 12, 15, 17)

[52] A. Rusbridger, The Snowden leaks and the public (2013). (Cited on page 12)

[53] Computerworld, Cloud computing 2014: Moving to a zero-trust security model, [Online]. Available from: http://www.computerworld.com/article/2487123/data-privacy/cloud-computing-2014--moving-to-a-zero-trust-security-model.html [Accessed 2015-05-04]. (Cited on page 12)

[54] The INQUIRER, Dropbox advises users with privacy concerns to add their own encryption, [Online]. Available from: http://www.theinquirer.net/inquirer/news/2356848/dropbox-advises-users-with-privacy-concerns-to-add-their-own-encryption [Accessed 2016-05-18]. (Cited on page 12)

[55] Lifehacker, Five Best Password Managers, [Online]. Available from: http://lifehacker.com/5529133/five-best-password-managers [Accessed 2015-12-31]. (Cited on page 13)

[56] J. Diaz, Seguridad en servicios de almacenamiento. Análisis de Dropbox y Mega, [Online]. Available from: https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/incibe_seguridad_almacenamiento_dropbox_mega.pdf [Accessed 2016-03-24]. (Cited on page 13)

[57] S. Josefsson, PKCS# 5: Password-Based Key Derivation Function 2 (PBKDF2) Test Vectors, Tech. rep. (2011). (Cited on page 13)

[58] PCWorld, The LastPass security breach: What you need to know, do, and watch out for, [Online]. Available from: http://www.pcworld.com/article/2936621/the-lastpass-security-breach-what-you-need-to-know-do-and-watch-out-for.html [Accessed 2015-12-31]. (Cited on page 13)

[59] W. Ford, J. Kaliski B.S., Server-assisted generation of a strong secret from a password, in: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on, 2000, pp. 176–180. (Cited on page 13)

[60] Dyadic, Protect the Keys to Everything — Dyadic Technology, [Online]. Available from: https://www.dyadicsec.com/technology/ [Accessed 2014-06-19]. (Cited on page 13)

[61] D. Bogdanov, S. Laur, J. Willemson, Sharemind: A framework for fast privacy-preserving computations, in: Computer Security-ESORICS 2008, Springer, 2008, pp. 192–206. (Cited on page 13)

[62] RSA Laboratories - EMC, What is a digital envelope?, [Online]. Available from: http://www.symmetrix.me/emc-plus/rsa-labs/standards-initiatives/what-is-a-digital-envelope.htm [Accessed 2016-05-02]. (Cited on page 14)

[63] P. K. Tysowski, Highly Scalable and Secure Mobile Applications in Cloud Computing Systems, Ph.D. thesis, University of Waterloo (2013). (Cited on page 14)

[64] K. Xue, P. Hong, A Dynamic Secure Group Sharing Framework in Public Cloud Computing, Cloud Computing, IEEE Transactions on 2 (4) (2014) 459–470. (Cited on page 15)

[65] A. Juels, B. S. Kaliski Jr, PORs: Proofs of retrievability for large files, in: Proceedings of the 14th ACM conference on Computer and communications security, 2007, pp. 584–597. (Cited on page 16, 170)

[66] D. Arroyo, J. Diaz, V. Gayoso, On the difficult tradeoff between security and privacy: challenges for the management of digital identities (JUNE) (2013) 2013. (Cited on page 16)

[67] G. Seroussi, I. Blake, N. Smart, Elliptic Curves in Cryptography, Cambridge University Press, 2000. (Cited on page 17, 102)

[68] H. Rifà-Pous, J. Herrera-Joancomartí, Computational and Energy Costs of Cryptographic Algorithms on Handheld Devices, Future Internet 3 (1) (2011) 31–48. (Cited on page 17)

[69] CloudFare, ECDSA: The digital signature algorithm of a better internet, [Online]. Available from: https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/ [Accessed 2016-03-23]. (Cited on page 17)

[70] N. Jansma, B. Arrendondo, Performance comparison of elliptic curve and RSA digital signatures, Technical Report, University of Michigan College of Engineering (2004) 1–20. (Cited on page 17)

[71] Crypto++Library, Speed Comparison of Popular Crypto Algorithms, [Online]. Available from: http://www.cryptopp.com/benchmarks.html [Accessed 2016-03-23]. (Cited on page 17)

[72] G. Becker, Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis, Ruhr-Universität Bochum, 2008. (Cited on page 18, 102)

[73] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, R. L. Rivest, How to tell if your cloud files are vulnerable to drive crashes, in: Proceedings of the 18th ACM conference on Computer and communications security, ACM, 2011, pp. 501–514. (Cited on page 18)

[74] Best Backups, 7 Cloud Storage Managers For Multiple Cloud Storage Services - Best Backups.com, [Online]. Available from: http://www.bestbackups.com/blog/4429/7-cloud-storage-managers-for-multiple-cloud-storage-services/ [Accessed 2016-03-26]. (Cited on page 18)

[75] H. S. Yeo, X. S. Phang, H. J. Lee, H. Lim, Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices, Journal of Network and Computer Applications 43 (2014) 142–156.
URL http://dx.doi.org/10.1016/j.jnca.2014.04.006 (Cited on page 18)

[76] The Guardian, What is 'safe harbour' and why did the EUCJ just declare it invalid?, [Online]. Available from: http://www.theguardian.com/technology/2015/oct/06/safe-harbour-european-court-declare-invalid-data-protection [Accessed 2016-01-08]. (Cited on page 19)

[77] C. Boyd, Cryptography in the Cloud: Advances and Challenges, Journal of Information and Communication Convergence Engineering 11 (1) (2013) 17–23. (Cited on page 19)

[78] P. Puzio, R. Molva, M. Onen, S. Loureiro, ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage, Proceedings of 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom'13) (2013) 363–370. (Cited on page 20, 22)

[79] EVault, Secure , Efficient Data Deduplication for Endpoint Protection, [Online]. Available from: https://www.evault.com/wp-content/uploads/2014/03/tech-brief_secure-deduplication-for-endpoints_us.pdf [Accessed 2015-10-31]. (Cited on page 20)

[80] J. Li, X. Chen, F. Xhafa, L. Barolli, Secure Deduplication Storage Systems with Keyword Search, Proceedings of 2014 IEEE 28th International Conference on Advanced Information Networking and Applications (AINA'14) (2014) 971–977. (Cited on page 20, 166)

[81] P. Puzio, R. Molva, M. Onen, S. Loureiro, PerfectDedup: Secure Data Deduplication 2020 (644412) 1–18. (Cited on page 21)

[82] M. Dworkin, R. Wilton, Recommendation for Block Cipher Modes of Operation X (December) (2009) 1–23. (Cited on page 22, 103)

[83] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, J. C. S. Lui, A secure cloud backup system with assured deletion and version control, Proceedings of the International Conference on Parallel Processing Workshops (2011) 160–167. (Cited on page 22, 23, 24)

[84] Dropbox, What happens when I delete a file? - Dropbox, [Online]. Available from: https://www.dropbox.com/help/115 [Accessed 2016-02-08]. (Cited on page 24)

[85] B. Butler, Researchers steal secret RSA encryption keys in Amazon's cloud, [Online]. Available from: http://www.networkworld.com/article/2989757/cloud-security/researchers-steal-secret-rsa-encryption-keys-in-amazon-s-cloud.html [Accessed 2015-11-22]. (Cited on page 24)

[86] Security Week, Microsoft Leaks User Account Identifiers in Clear Text, [Online]. Available from: http://www.securityweek.com/microsoft-leaks-user-account-identifiers-clear-text [Accessed 2015-11-22]. (Cited on page 25)

[87] J. Ransome, A. Misra, Core Software Security: Security at the Source, CRC Press, 2013. (Cited on page 25, 169)

[88] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, S. Maffeis, Discovering concrete attacks on website authorization by formal analysis, Journal of Computer Security 22 (4) (2014) 601–657. (Cited on page 25)

[89] J. Diaz, D. Arroyo, F. B. Rodriguez, A formal methodology for integral security design and verification of network protocols, The Journal of Systems and Software. (Cited on page 25)

[90] C. Meadows, Emerging Issues and Trends in Formal Methods in Cryptographic Protocol Analysis: Twelve Years Later, in: Logic, Rewriting, and Concurrency, Springer, 2015, pp. 475–492. (Cited on page 26, 103)

[91] A. Whitten, J. D. Tygar, Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0., in: Usenix Security, Vol. 1999, 1999. (Cited on page 26)

[92] K. Renaud, M. Volkamer, A. Renkema-Padmos, Why doesn't jane protect her privacy?, in: Privacy Enhancing Technologies, Springer, 2014, pp. 244–262. (Cited on page 26)

[93] A. Cavoukian, M. Dixon, Privacy and Security by Design: An Enterprise Architecture Approach, Ontario: Information and Privacy Commissioner, 2013. (Cited on page 26)

[94] CERT Secure Coding Standards, Top 10 Secure Coding Practices, [Online]. Available from: https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices [Accessed 2016-05-22]. (Cited on page 26)

[95] C.-M. Karat, C. Brodie, J. Karat, Usability design and evaluation for privacy and security solutions, Security and Usability (2005) 47–74. (Cited on page 27)

[96] K. Radke, C. Boyd, J. G. Nieto, M. Brereton, Ceremony analysis: Strengths and weaknesses, in: Future Challenges in Security and Privacy for Academia and Industry, Springer, 2011, pp. 104–115. (Cited on page 27)

[97] K. Radke, C. Boyd, J. G. Nieto, H. Bartlett, CHURNs: freshness assurance for humans, The Computer Journal (2014) bxu073. (Cited on page 27)

[98] E. B. Barker, W. C. Barker, W. E. Burr, W. T. Polk, M. E. Smid, SP 800-57. Recommendation for Key Management, Part 1: General (Revised), Tech. rep. (2016).
URL http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf (Cited on page 46)

[99] B. Madhuravani, D. S. R. Murthy, Cryptographic Hash Functions : SHA Family (4) (2013) 326–329. (Cited on page 47)

[100] Microsoft, Hard disk UUID, [Online]. Available from: https://social.technet.microsoft.com/Forums/windowsserver/en-US/fb0d42af-e6bf-490d-860c-3641c22931dc/hard-disk-uuid?forum=winservergen [Accessed 2016-05-31]. (Cited on page 103)

[101] Archlinux, Persistent block device naming, [Online]. Available from: https://wiki.archlinux.org/index.php/persistent_block_device_naming [Accessed 2016-05-31]. (Cited on page 103)

[102] M. Rouse, What is Advanced Encryption Standard (AES)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard [Accessed 2016-05-02]. (Cited on page 165)

[103] M. Rouse, What is application program interface (API)?, [Online]. Available from: http://searchexchange.techtarget.com/definition/application-program-interface [Accessed 2016-05-02]. (Cited on page 165)

[104] M. Rouse, What is cipher block chaining (CBC)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/cipher-block-chaining [Accessed 2016-05-31]. (Cited on page 165)

[105] M. Rouse, What is cloud access security broker (CASB)?, [Online]. Available from: http://searchcloudsecurity.techtarget.com/definition/cloud-access-security-brokers-CABs [Accessed 2016-05-30]. (Cited on page 165)

[106] Usability Body of Knowledge, Cognitive Walkthrough, [Online]. Available from: http://www.usabilitybok.org/cognitive-walkthrough [Accessed 2016-05-02]. (Cited on page 166)

[107] M. Rouse, What is cyclic redundancy checking?, [Online]. Available from: http://searchnetworking.techtarget.com/definition/cyclic-redundancy-checking [Accessed 2016-05-30]. (Cited on page 166)

[108] M. Rouse, What is Electronic Code Book (ECB)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/Electronic-Code-Book [Accessed 2016-05-31]. (Cited on page 166)

[109] M. Rouse, What is end-to-end encryption (E2EE)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/end-to-end-encryption-E2EE [Accessed 2016-05-30]. (Cited on page 166)

[110] J. Nielsen, Usability engineering, Elsevier, 1994. (Cited on page 167, 171)

[111] UsabilityNet, Heuristic evaluation, [Online]. Available from: http://www.usabilitynet.org/tools/expertheuristic.htm [Accessed 2016-05-24]. (Cited on page 167)

[112] M. Rouse, What is HCI (human-computer interaction)?, [Online]. Available from: http://searchsoftwarequality.techtarget.com/definition/HCI-human-computer-interaction [Accessed 2016-05-30]. (Cited on page 167)

[113] Techopedia, What is Information and Communications Technology (ICT)?, [Online]. Available from: https://www.techopedia.com/definition/24152/information-and-communications-technology-ict [Accessed 2016-05-02]. (Cited on page 167)

[114] M. Hamizi, Cloud Computing : SPI model (SaaS, PaaS, IaaS), [Online]. Available from: https://mizitechinfo.wordpress.com/2013/10/12/cloud-computing-spi-model-saas-paas-iaas-part-17/ [Accessed 2016-05-30]. (Cited on page 167, 169, 170, 171)

[115] Microsoft Research, Multi-Party Computation, [Online]. Available from: http://research.microsoft.com/en-us/projects/mpc/ [Accessed 2016-05-24]. (Cited on page 167)

[116] Techopedia, What is OAuth?, [Online]. Available from: https://www.techopedia.com/definition/26694/oauth [Accessed 2016-05-30]. (Cited on page 168)

[117] M. Rouse, What is one-time password (OTP)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/one-time-password-OTP [Accessed 2016-05-24]. (Cited on page 168)

[118] M. Rouse, What is open source?, [Online]. Available from: http://searchenterpriselinux.techtarget.com/definition/open-source [Accessed 2016-05-02]. (Cited on page 168)

[119] M. Rouse, What is output feedback (OFB)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/output-feedback [Accessed 2016-05-31]. (Cited on page 168)

[120] M. Rouse, What is personally identifiable information (PII)?, [Online]. Available from: http://searchfinancialsecurity.techtarget.com/definition/personally-identifiable-information [Accessed 2016-05-30]. (Cited on page 168)

[121] M. Rouse, What is Pretty Good Privacy (PGP)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/Pretty-Good-Privacy [Accessed 2016-05-24. (Cited on page 169)

[122] Techopedia, What is RSA Encryption?, [Online]. Available from: https://www.techopedia.com/definition/21852/rsa-encryption [Accessed 2016-05-02]. (Cited on page 169)

[123] M. Rouse, What is single point of failure (SPOF)?, [Online]. Available from: http://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF [Accessed 2016-05-30]. (Cited on page 170)

[124] M. Rouse, What is single sign-on (SSO)?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/single-sign-on [Accessed 2016-05-30]. (Cited on page 170)

[125] The FSE Group, Definition of an SME, [Online]. Available from: http://www.thefsegroup.com/definition-of-an-sme/ [Accessed 2016-05-25]. (Cited on page 170)

[126] M. Rouse, What is systems development life cycle (SDLC)?, [Online]. Available from: http://searchsoftwarequality.techtarget.com/definition/systems-development-life-cycle [Accessed 2016-05-30]. (Cited on page 170)

[127] M. Rouse, What is two-step verification?, [Online]. Available from: http://searchsecurity.techtarget.com/definition/two-step-verification [Accessed 2016-05-30]. (Cited on page 171)

Glossary

Advanced Encryption Standard (AES) Symmetric block cipher used to encrypt sensitive data [102]. 46–48, 59, 61, 63, 74, 103

Application Program Interface (API) Code that allows two software programs to communicate with each other [103]. 2, 24–26, 31, 45

Cipher Block Chaining (CBC) Mode of operation for a block cipher (one in which a sequence of bits are encrypted as a single unit or block with a cipher key applied to the entire block). Cipher Block Chaining uses what is known as an initialisation vector (IV) of a certain length. One of its key characteristics is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks. As a result, the entire validity of all preceding blocks is contained in the immediately previous ciphertext block. A single bit error in a ciphertext block affects the decryption of all subsequent blocks. Rearrangement of the order of the ciphertext blocks causes decryption to become corrupted. Basically, in cipher block chaining, each plaintext block is XORed (see XOR) with the immediately previous ciphertext block, and then encrypted [104]. 22, 46–48, 61, 74, 103

Cloud Access Security Broker (CASB) Software tool or service that sits between an organisation's on-premises infrastructure and a cloud provider's infrastructure. A CASB acts as a gatekeeper, allowing the organisation to extend the reach of their security policies beyond their own infrastructure [105]. 10

Cloud Provider (CP) Cloud service provider. 1–4, 8, 10–12, 14–16, 18–20, 23, 24, 29, 30, 59, 72

Cognitive walkthrough Usability evaluation method in which one or more evaluators work through a series of tasks and ask a set of questions from the perspective of the user [106]. 26

Convergent Encryption (CE) Technique that consists of using as encryption key the hash of the content of data we want to encrypt, such that two equal files will generate the same encrypted file [80]. 20–22

Cyclic Redundancy Checking (CRC) Method of checking for errors in data that has been transmitted on a communications link [107]. 16

Dynamic Authorisation Management (DAM) Mechanisms which consist of externalising authorization decisions outside of applications, using an "application security infrastructure" which performs these authorization decisions [39]. 10

Electronic Code Book (ECB) Mode of operation for a block cipher, with the characteristic that each possible block of plaintext has a defined corresponding ciphertext value and vice versa. In other words, the same plaintext value will always result in the same ciphertext value. Electronic Code Book is used when a volume of plaintext is separated into several blocks of data, each of which is then encrypted independently of other blocks. In fact, Electronic Code Book has the ability to support a separate encryption key for each block type [108]. 46, 103

End-to-End Encryption (E2EE) Method of secure communication that prevents third-parties from accessing data while it is transferred from one end system or device to another [109]. 12

FCS Free Cloud Storage. 11, 18, 29, 30, 101

Focus Group Technique that brings together from six to nine users to discuss issues and concerns about the features of a user interface. The group typically lasts about two hours and is run by a moderator who maintains the group's focus [110]. 104

Heuristic evaluation Usability inspection where usability specialists judge whether each element of a user interface follows a list of established usability heuristics [111]. 26

Human-Computer Interaction (HCI) Study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings [112]. 27

Information and Communications Technology (ICT) Refers to all the technology used to handle telecommunications, broadcast media, intelligent building management systems, audiovisual processing and transmission systems, and network-based control and monitoring functions. [113]. 1, 2, 5, 7, 15

Infrastructure As A Service (IaaS) Enables users to rent equipment and run a data center. In an IaaS arrangement, clients are typically billed based on the resources they consume [114]. 6

Multi-party computation (MPC) Protocol that allows a set of parties, each with a private input, to securely and jointly perform any computation over their inputs [115]. 13

NPM Package manager for JavaScript that allows to find, share, and reuse packages of code from hundreds of thousands of developers and assemble them in powerful new ways. 43

OAuth Authorization protocol that allows a third-party website or application to access a user's data without the user needing to share login credentials [116]. 9, 10

One-time password (OTP) Automatically generated numeric or alphanumeric string of characters that authenticates the user for a single transaction or session [117]. 13

Open Source Program whose source code is made available for use or modification as users or other developers see fit. Open Source software is usually developed as a public collaboration and made freely available [118]. i, ii, 2, 3, 25, 29, 30, 101

Output Feedback (OFB) Mode of operation for a block cipher. It permits encryption of differing block sizes and the output of the encryption block function is the feedback. The XOR (exclusive OR) value of each plaintext block is created independently of both the plaintext and ciphertext. It is this mode that is used when there can be no tolerance for error propagation, as there are no chaining dependencies. It uses an initialisation vector (IV). Changing the IV in the same plaintext block results in different ciphertext [119]. 103

Password-based Authenticated Key Exchange (PAKE) Particular case of authenticated key exchange protocols in which the secret key or password used for authentication is not uniformly distributed over a large space, but rather chosen from a small set of possible values (a four-digit pin, for example) [31]. 9

Personally Identifiable Information (PII) Any data that could potentially identify a specific individual. Any information that can be used to distinguish one person from another and can be used for de-anonymising anonymous data can be considered PII [120]. 10

Platform As A Service (PaaS) Platform that delivers a solution stack as a service. It supports in deployment of applications. It does not involve the cost and complexity of buying and managing the underlying hardware and software and even the provisioning hosting capabilities. It provides all the facilities that are required to support the complete life cycle of building and delivering web applications and services, which are completely accessible from the Internet [114]. 6

Pretty Good Privacy (PGP) Popular program used to encrypt and decrypt email over the Internet, as well as authenticate messages with digital signatures and encrypted stored files [121]. 26

Rivest-Shamir-Adleman (RSA) RSA encryption is a public-key encryption technology which is based on the difficulty in factoring very large numbers. Based on this principle, the RSA encryption algorithm uses prime factorization as the trap door for encryption. Deducing an RSA key, therefore, takes a huge amount of time and processing power. RSA is the standard encryption method for important data, especially data that is transmitted over the Internet [122]. 17, 37, 38, 44, 46, 48–50, 55–58, 74, 102

Secure Development Lifecycle (SDL) Methodology that provides best practices for developing secure applications, with the aim of overcoming the challenges of making software secure [87]. 25

Secure Hash Algorithm-256 (SHA-256) Cryptographic hash function with digest length of 256 bits. 47

Secure Remote Password (SRP) Secure password-based authentication and key-exchange protocol. It solves the problem of authenticating clients to servers securely, in cases where the user of the client software must memorize a small

secret (like a password) and carries no other secret information, and where the server carries a verifier for each user, which allows it to authenticate the client but which, if compromised, would not allow the attacker to impersonate the client [32]. 9

Sentinels Set of randomly-valued check blocks [65]. 16

Single Point of Failure (SPOF) Potential risk posed by a flaw in the design, implementation or configuration of a circuit or system in which one fault or malfunction causes an entire system to stop operating [123]. 13

Single Sign-On (SSO) User authentication process that permits a user to enter one name and password in order to access multiple applications [124]. 11

Small or Medium-sized Enterprise (SME) Business or company that has fewer than 250 employees; and has either (a) annual turnover not exceeding €50 million (approximately £40 million) or (b) an annual balance-sheet total not exceeding €43 million (approximately £34 million) [125]. ii, 1, 4, 6, 30

Software As A Service (SaaS) Cloud implementation that delivers software or, more generally described, an application to its end user. SaaS enables a service provider to make applications available over the Internet. This capability eliminates the need to install software on users' computers, and it can be helpful for mobile or transient workforces [114]. 6

Software Development Life Cycle (SDLC) Conceptual model used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application [126]. 25, 27

Software, Platform, Infrastructure (SPI) model Term that encompasses three

popular types of cloud computing services: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [114]. 6

SP Service Provider. 1, 5, 6, 11, 12

Thinking aloud Test in which is asked participants to use the system while continuously thinking out loud, that is, simply verbalizing their thoughts as they move through the user interface [110]. 104

Two-Step Verification (2SV) Process that involves two authentication methods performed one after the other to verify that someone or something requesting access is who or what they are declared to be [127]. 9–11, 103