MS ENGG. PROJECT

A Prototype of a Secured File Storing and Sharing System for Cloud Storage Infrastructure

Submitted in partial fulfillment of
the requirements for the award of the degree of

Master of Science
in
Computer Science and Engineering

Submitted by

Md. Shahadul Alam Patwary

ID: 1815414050

Under the Supervision of

Dr. Rajesh Palit

Department of Electrical & Computer Engineering
North South University

December 2019

DECLARATION

I hereby, declare that the work presented in this report is the outcome of the design and development work performed by myself, Md. Shahadul Alam Patwary, under the supervision of Dr. Rajesh Palit, Associate Professor, Department of Electrical & Computer Engineering, North South University as a course work of CSE 590 (Final Project). I also declare that no part of this report has been taken from other works without reference.

Signature

Md. Shahadul Alam Patwary
ID: 1815414050

APPROVAL

This project report titled ‘A Prototype of a Secured File Storing and Sharing System for Cloud Storage Infrastructure’ submitted by MD. SHAHADUL ALAM PATWARY, ID: 1815414050 to the department of Electrical & Computer Engineering, North South University, has been accepted as the final term project report.

Signatures

Dr. Rajesh Palit
Associate Professor, Department of ECE
Superviser

Dr. Ashrafuzzaman Khan
Asst. Professor, Department of ECE
Committee Member (Internal)

Dr. Salekul Islam
Professor & Head
Department of Computer Science and Engineering
United International University
Committee Member (External)

Dr. K. M. A. Salam
Professor & Chairman
Department of ECE

Table of contents

Abstract

With the increased popularity of third-party cloud storage services, privacy has become a major concern. As cloud storage services can be availed at a minimum price now-a-days, it comes with a greater cost, vulnerability of being accessed by anyone else. Most of the cloud service providers keep multiple copies of files to ensure reliability to their users. But this turns into a serious privacy issue when the replicated copies remain hidden in the server even after the owner has deleted the original file.

To solve this problem, several models were proposed that ensures assured file deletion. Some of them work by imposing policies on files stored in the cloud e.g. FADE, some are based on Attribute Based Encryption e.g. the model proposed by Bentajer et al. (2019), some even proposed self destructive documents which relies on expiration time etc. Other schemes worth mentioning e.g. MADS, SFADE, SFADE+ etc. all of them are based on standard cryptographic techniques and provide privacy and security to the files stored in the cloud. But all these models have shortcomings of their own. e.g. FADE comes with a complex architecture, relies on a key manager and also has design vulnerability. SFADE then solves these issues but is limited to file uploading and downloading. To enable sharing, an incremental improvement was made to the SFADE architecture and a new method, SFADE+ was proposed but it became computationally heavy.

So we have designed a system using the existing cryptographic approaches which ensures assured deletion of files, resolving all the limitations that remained unpatched by the previous models. The primary focus of our proposed model is to ensure user privacy and security through a user-friendly system which is less computationally intensive and does not rely upon any trusted entity. We also have one significant enhancement to offer over SFADE/SFADE+ by providing recovery option for forgotten passphrases.

For the demonstration of our proposed model, we have implemented a prototype named CryptedCloud as an overlay system on top of Google Drive. It factually exhibits that CryptedCloud assures file deletion without relying upon any trusted body with no compromise to the performance and user-friendliness.

1 Introduction

The idea of network based computing has been around since 1960 but around 2006, cloud storage services became mainstream with the introduction of cloud storage service AWS S3 by Amazon Web Services. Cloud storage services provide infinite amount of storage, makes it easier to access files from anywhere around the world with minimum data management cost.

1.1 Cloud Computing and Cloud Infrastructure

According to a case study done by Yamasaki et al. (2015) [1] on IaaS and SaaS in Public Cloud, three kinds of services can be provided through cloud computing such as; Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS).

Software as a service (SaaS) is a software delivery model in which applications are hosted by third-party providers who make those applications available to customers over the internet. The consumer only needs to make changes to some application-specific configurations and manage users and the rest of the components such as; all of the infrastructure, all of the application logic, all deployments, and everything related to the delivery of the product or service will be handled by the service provider [2]. A monthly subscription fee is usually associated with SaaS. One of the biggest advantages of the SaaS model is that downloading software on multiple computers and keeping them up-to-date on each of them no longer requires engagement of IT specialists. As everything comes for a cost, SaaS is no different than that. The biggest disadvantage of SaaS is that total control is handed over to the third party who cannot be relied upon for managing critical business applications.

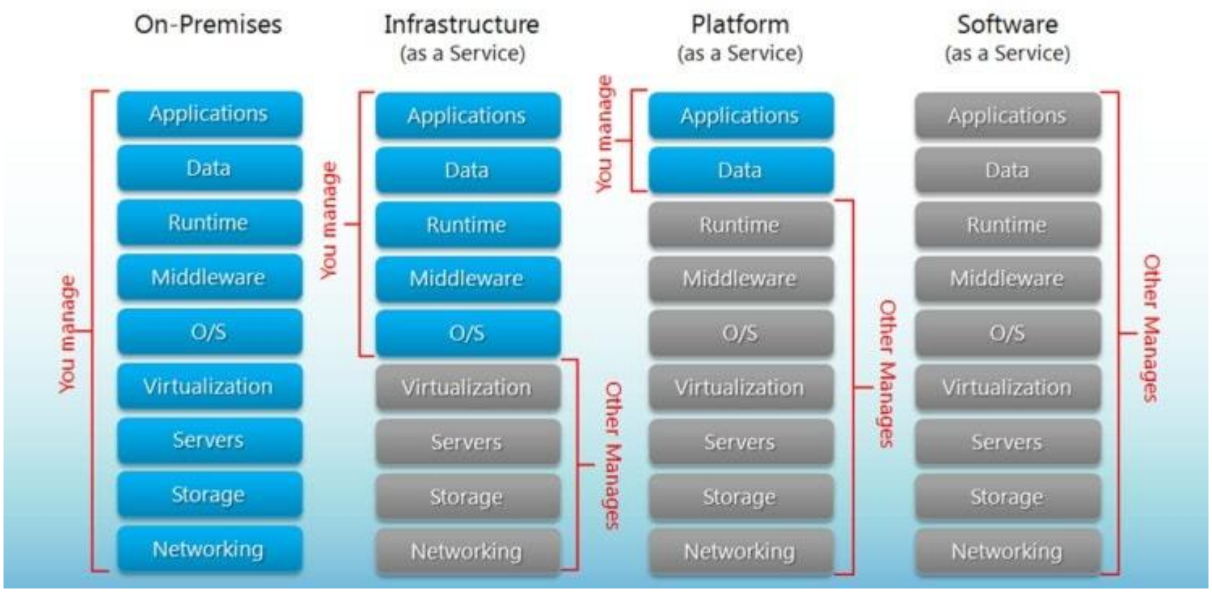


Figure 1.1: SaaS vs. PaaS vs. IaaS

Platform-as-a-Service (PaaS) provides application development tools and its customer base is mostly developers. According to an article named “PaaS: New Opportunities for Cloud Application Development” by Cohen, B. (2013) [3], tools provided by PaaS not only manage development but they also organize the complete Software Development Life Cycle. He also mentioned that, PaaS will become more widely established in larger corporate enterprises as the environments and tools mature. One of the key advantages of PaaS is that developers do not need to write applications from scratch which saves them a lot of time and money on writing extensive code. But it comes with a cost of data security as information is stored off-site.

Infrastructure-as-a-Service (IaaS) offers services such as storage, networking, and virtualization in a pay-as-you-go manner. It provides cloud-based alternatives to in-house hardware and software components (e.g. storage, virtualization software etc.) to users which reduces investment in expensive in-house resources. One key benefit of IaaS is that it can be purchased as per the need of the customer. As IaaS model is highly flexible and scalable, more of IaaS service can be bought as business grows. One remarkable draw-back of IaaS is that it highly depends upon virtualization services. It also restricts users' privacy & customization.

In addition to that, there is another kind of service that can be delivered over cloud; known as Storage-as-a-Service. It is a business model in which a company provides its storage infrastructure to another company or individuals to store data. There are many advantages of this cloud service; such as, it reduces the cost associated with traditional backup methods, provides an abstraction of infinite storage space and many more. While this service is availed, the primary concern would be privacy and security.

Cloud computing does not refer to a single thing instead; it describes a variety of individual components or services. The combination of different models of cloud computing are referred to as Cloud stack.

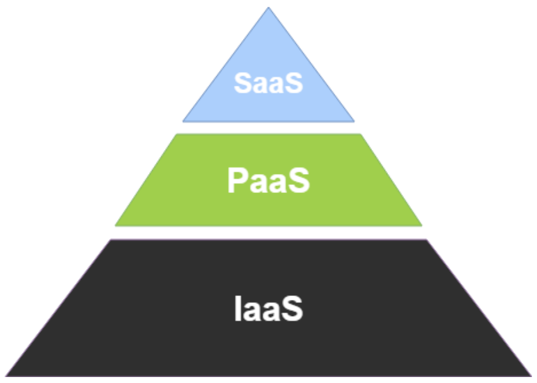


Figure 1.2: Cloud stack

1.2 What is Assured File Deletion?

With all these cloud based service models, there is one common downside, lack of privacy and security. Although cloud storage services provide us *integrity* and *availability*, but it lacks to provide us *confidentiality*. As data is stored on a centralized location known as data centers having a large size of data storage, and the data being processed somewhere on servers, clients need to trust the provider on the availability and confidentiality. And the only legal agreement between the service provider and client is the Software License Agreement (SLA) [4]. But at times, even SLA might not be very helpful as there is no defined standard for cloud computing. Unexpected violation of any of the terms mentioned in SLA by service provider might cause serious privacy issue. As cloud service providers store replicated files to prevent unexpected data-loss, even after deletion of data, replicated data still remains. But in some scenarios, it becomes crucial to destroy files in a way that nobody can recover. e.g. a software company stores their source code in a third party cloud storage. After a couple of years, if they switch to another cloud storage service, it is highly probable that they would want to delete the source code stored in previous cloud storage and it should be unrecoverable.

1.3 Existing Models and Limitations

To ensure such privacy, a model was proposed by Tang Y., Lee P.P.C., Lui J.C.S., Perlman R. in their paper FADE: Secure Overlay Cloud Storage with File Assured Deletion [5] published in 2010, which uses policy-based file assured deletion to restrict unwanted access to data stored in 3rd party cloud storage. FADE assures that a file becomes un-recoverable after the access to that particular file has been revoked. There are certain limitations and vulnerabilities of FADE architecture. e.g. using FADE system, a file can be accessed by a specified group but it has no method defined for explicitly sharing a file with others.

A more simplified model SFADE was proposed by Habib et al. (2013) [6]. It does not require any 3rd party key manager. A randomly generated key is used for encrypting files and that key is then encrypted by a secret key generated using user’s passphrase. But SFADE lacks file sharing support.

In 2017, an extended version of SFADE named SFADE+ was proposed by Nusrat et al. (2017) [7] which supports file sharing. It uses RSA algorithm to securely share files. Private key is used for file encryption. As it is generated based on user’s passphrase, it is not stored anywhere. Public key is used for decryption and it is stored in a repository along with the location of the files to be stored. SFADE+ overcomes the limitation of its predecessor, SFADE; but it comes with its own shortcomings. e.g. SFADE+ adds an extra layer of computational overhead due to RSA key generation for uploading each file.

1.4 Our Proposed Model

As all these models for assured file deletion has their own potentials and drawbacks, we have designed a secure system architecture which resolves most of the limitations of previously proposed models keeping the potentials of its predecessors intact. Our model consists of two components, a client and a server. All the cryptographic operations are performed by the client application. The only task for server is to store some information and deliver them to the client when needed. The primary goals of our designed model are, improved file sharing support, reduced computational complexity, enhanced security and removing the necessity of the server side being trusted entity. Our model also supports passphrase recovery as forgetting passphrase is very common for users and it is a frustrating experience.

File sharing support was improved significantly as it delivers complete control over file access to the owner of the file. A friendly user interface was designed to ensure uncompromised file sharing experience.

In order to share files, our model also relies upon RSA algorithm just like SFADE+ [7]. But our proposed model is designed in such a way that, RSA key generation is done only once for each user and RSA decryption is also done only once per user throughout the lifecycle of a file which reduces the load from the CPU of the user's device and increases the performance of our model significantly.

Though our model relies on an additional server, privacy is not in the hands of the server administrator. It is due to the fact that the server does not contain any information which can be used by anyone else other than the user.

1.5 Organization of the Report

We will have an in-depth discussion regarding how all the operations in FADE, SFADE and SFADE+ works and their limitations in Chapter 2. In Chapter 3, we will discuss how our proposed model works and why it is more secure and how it prevails over the shortcomings of its predecessors. In Chapter 4, we will discuss the implementation details of our proposed model. In Chapter 5, we will discuss the limitations of our system. We will talk about our future plans with this project in Chapter 6.

2 Literature Review

At first, we will discuss how FADE [5] works. As we have mentioned earlier, FADE assures file deletion by applying policies on files. There are three participants in this system; i) Data owner who uploaded the file to the cloud storage, ii) Trusted key manager which maintains the policy based control keys that are used to encrypt data keys. It performs necessary operations (e.g. encryption, decryption, renewal etc.) to control keys according to data owner's request, iii) Cloud storage where the files are stored.

For upload operation, data owner needs to request the public key of policy P_i from the key manager. Then the data owner needs to generate two random keys K and S_i, and stores K encrypted by S_i, $\{K\}_{Si}$; S_i encrypted by e_i, S_i^{ei} and file encrypted by K, $\{F\}_K$ in the cloud storage. Then K and S_i can be discarded by the data owner. Figure 2.1 resembles file upload operation in FADE.

For download operation, data owner needs to retrieve $\{K\}_{Si}$ S_i^{ei} and $\{F\}_K$ from the cloud storage. Then a secret random number R is generated and R^{ei} is computed by the data owner. $S_i^{ei} \cdot R^{ei} = (S_iR)^{ei}$ is then sent to the key manager for decryption. The key manager computes and returns $((S_iR)^{ei})^{di} = S_iR$ to the data owner. Now S_i can be obtained and $\{K\}_{Si}$ can be decrypted and so as the $\{F\}_K$. Figure 2.2 describes the file download operation in FADE.

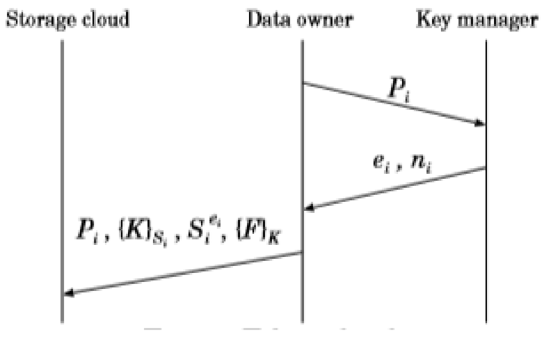


Figure 2.1: File upload in FADE [5]

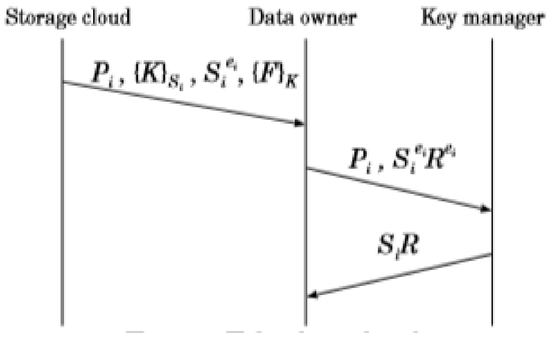


Figure 2.2: File download in FADE [5]

Now we will discuss some of the limitations of FADE architecture. First of all, it relies upon a 3rd party key manager which is assumed to be trusted. If key manager collaborates with 3rd party cloud storage provider, files can be decrypted easily. Secondly, this architecture supports accessing a file by a group of users but it has no method defined for explicitly sharing a file with another user or multiple specified users. Another issue with FADE architecture is that it is relatively complex in terms of storing the keys using key managers [6]. Alongside limitations, there are few vulnerabilities which makes FADE system architecture insecure. First of all, policy (P_i) is sent to cloud or key manager in unencrypted or plaintext form. So, during upload operation, if an attacker gets P_i by sniffing, attacker can send P_i to key manager and key manager will send corresponding public key to attacker and attacker can redirect to client. So, attacker can get P_i and public key (e_i, n_i) of policy easily. Then during download operation, if attacker gets encrypted secret key S_i^{ei} by sniffing, attacker can generate random number and send with S_i^{ei}, R^{ei} and P_i to key manager. Key manager will send the secret key to the attacker and the attacker will be able to decrypt the file [8].

Now we will talk about a more simplified approach called SFADE [6]. For upload operation in SFADE, a secret passphrase (Phrase1) needs to be entered by the user. SFADE then generates a secret key (Key2). The length of the key will be 64 bit or 128 bit. Key2 is used to encrypt the data/file and using Phrase 1, another key (Key1) is generated which is used to encrypt Key2. Key1 is derived from Phrase1 using SHA2 algorithm. Both encrypted Key2 and encrypted file are uploaded to the cloud. Figure 2.3 describes the file upload operation in SFADE.

For downloading a file, user must enter the secret passphrase (Phrase1) and Key1 is generated. Key1 is used to decrypt the encrypted Key2. Key2 is then used to decrypt the file and the file is downloaded. Figure 2.4 shows the file download operation in SFADE.

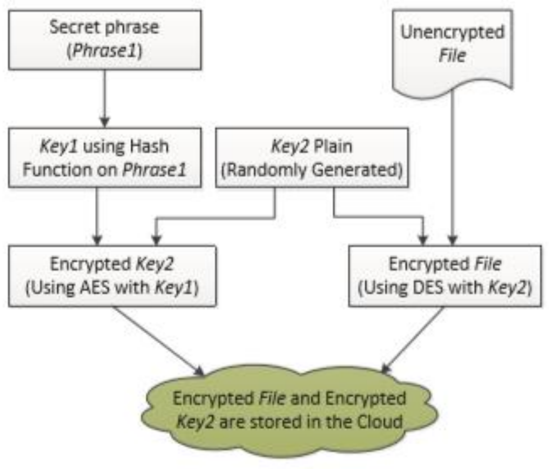

Secret phrase (*Phrase1*)
Unencrypted *File*
Key1 using Hash Function on *Phrase1*
Key2 Plain (Randomly Generated)
Encrypted *Key2* (Using AES with *Key1*)
Encrypted *File* (Using DES with *Key2*)
Encrypted *File* and Encrypted *Key2* are stored in the Cloud

Figure 2.3: File upload in SFADE [6]

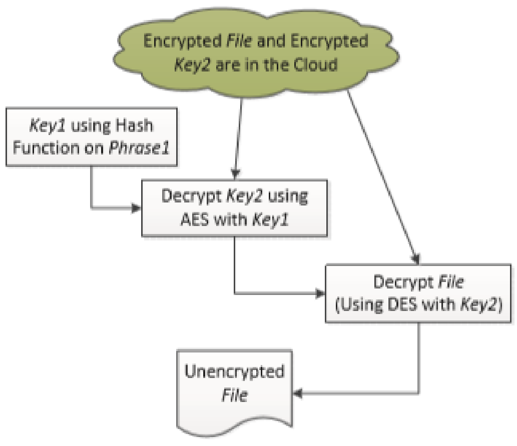

Encrypted *File* and Encrypted *Key2* are in the Cloud
Key1 using Hash Function on *Phrase1*
Decrypt *Key2* using AES with *Key1*
Decrypt *File* (Using DES with *Key2*)
Unencrypted *File*

Figure 2.4: File download in SFADE [6]

Now let us talk about some limitations in SFADE architecture. As mentioned previously, the most prominent drawback of SFADE is that it does not support file sharing. Suppose Alice wants to share a file with Bob and nobody other than Alice and Bob can access the file. But using SFADE, this cannot be achieved. Secondly, as DES is used for file encryption/decryption, it can be broken easily with brute-force attack. Thirdly, if user forgets the passphrase, there is no way to recover the file. Another minor performance overhead is hashing user's passphrase which could have been minimized by defining a minimum length for passphrase.

We will now talk about how the extended version of SFADE named as SFADE+ [7] works and how it overcomes the limitations of SFADE and introduces some of its own drawbacks.
At first, SFADE+ has its own registration process. It will generate RSA private-public key pair. All users need to go through the registration process. Figure 2.5 shows the user registration in SFADE+.

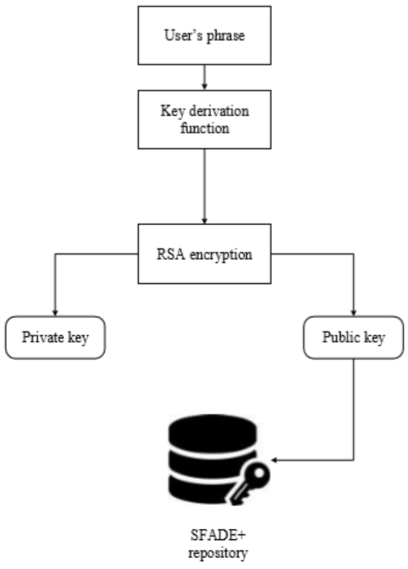
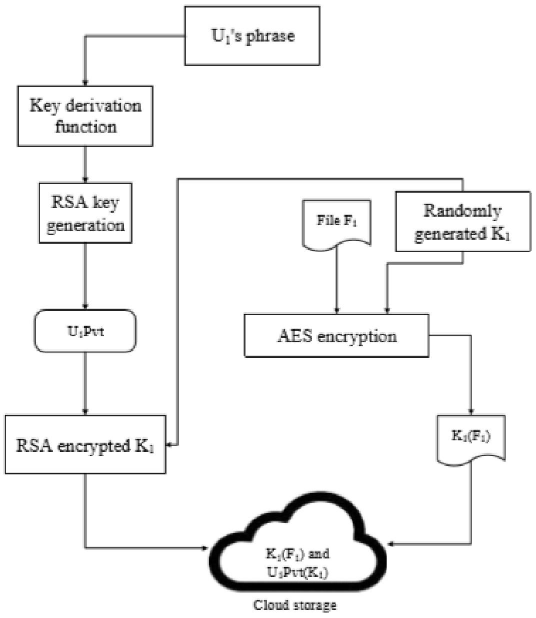


Figure 2.5: User registration process of SFADE+ [7]

Uploading a file F_1 through SFADE+ client works in a way that, F_1 is encrypted by a random key K_1 using AES. Then user is asked to enter the passphrase which then generates RSA private key U_1Pvt. SFADE+ generates a random key K_1 which encrypts F_1 using AES. U_1Pvt is used to encrypt K_1. Encrypted F_1, $K_1(F_1)$ and encrypted K_1, $U_1Pvt(K_1)$ are then uploaded to the cloud. Figure 2.6 shows the file upload operation in SFADE+.



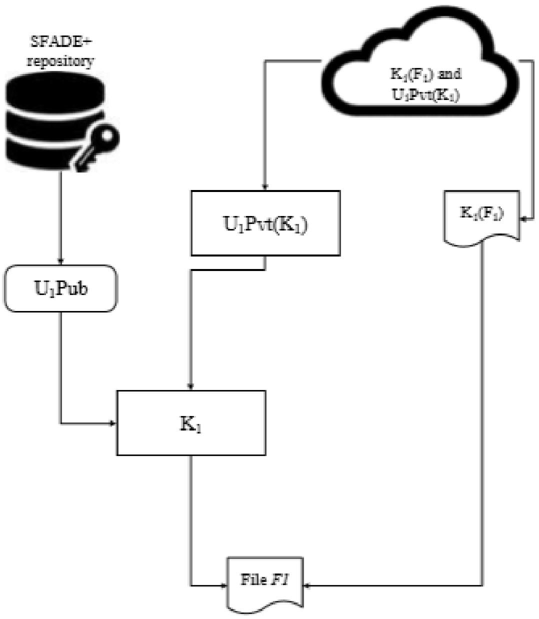


Figure 2.6: File upload in SFADE+ [7]   Figure 2.7: File download in SFADE+ [7]

For downloading a file F_1 in SFADE+, user needs to enter the passphrase. Then using the user's public key U_1Pub of the key pair, encrypted K_1 is decrypted which will be used to decrypt the encrypted F_1. The download mechanism of SFADE+ is shown in Figure 2.7.

If user 1 wants to share a file F_1 with user 2, user 1 needs to provide user 2's identity. SFADE+ will search for user 2 in the repository and retrieve user 2's public key, U_2Pub. Data key K_1 is taken and encrypted with U_2Pub by User 1's SFADE+ client and resulting in $U_2Pub(K_1)$. User 1's SFADE+ client will upload $U_2Pub(K_1)$ and collect file link of $K_1(F_1)$ and $U_2Pub(K_1)$ in the cloud. The file share mechanism of SFADE+ is shown in Figure 2.8.

When user 2 want to receive the file F_1 shared by user 1, he/she needs to provide SFADE+ with the passphrase. The private key, U_2Pvt derived from the passphrase is used to decrypt K_1 and K_1 is used to decrypt F_1. User 2 may re-encrypt and store K_1 in his/her cloud storage for future use. Figure 2.9 shows the file receive operation in SFADE+.

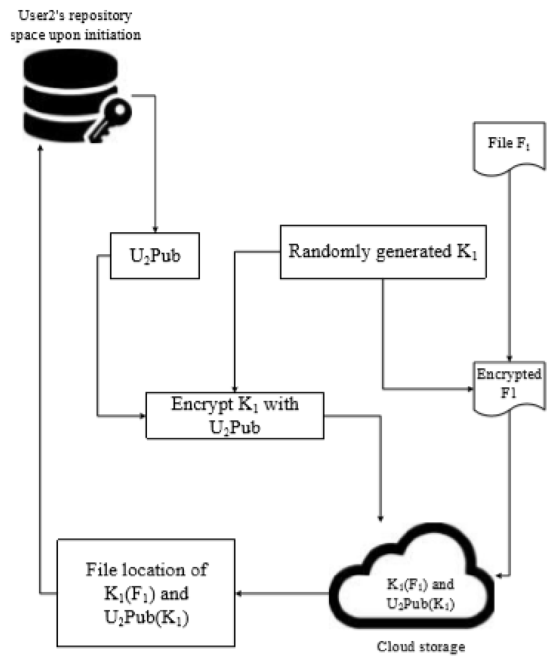


Figure 2.8: File share in SFADE+ [7]

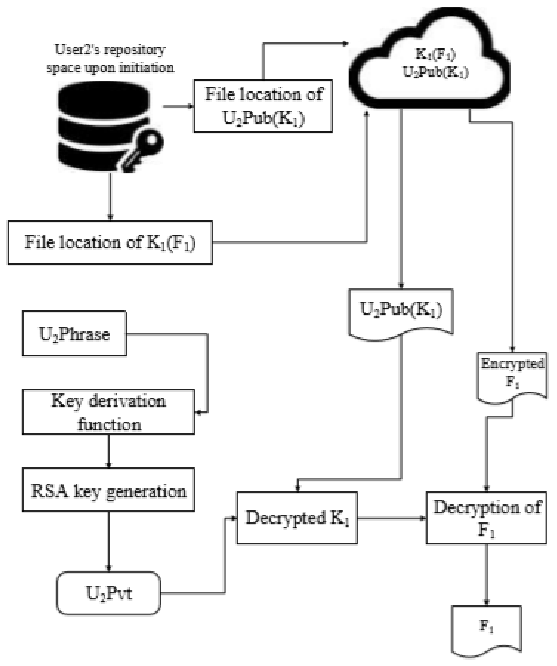


Figure 2.9: File receive in SFADE+ [7]

Now we will discuss some of the downsides of SFADE+ architecture. First and foremost, as mentioned earlier, there is an additional layer of computational overhead due to RSA key generation for uploading each individual file as private key is not stored anywhere and needs to be generated from the user's passphrase. Secondly, as most of the cloud service APIs do not provide direct access to the file location, it is impractical approach to store file location in repository. Thirdly, just like its predecessor SFADE, there is no way to recover the file for forgotten passphrase.

In 2016, a multi-replica associated deleting scheme (MADS) in a cloud environment was proposed by Zhang et al. (2016) [9] that ensures the privacy of the data and allows deletion of any associated replica. In this scheme, when a Replica Associated Object (RAO) generated from the source data is received, a cloud server records the replica information of that RAO which follows the multi-replica correlation mechanism. The replica correlation model is then established synchronously.

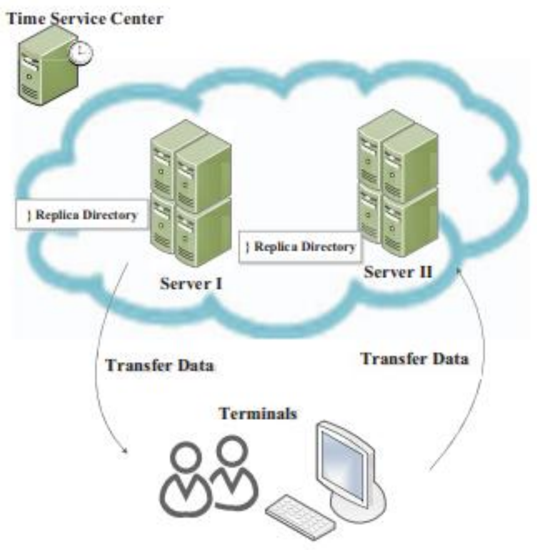


Figure 2.10: System model of MADS [9]

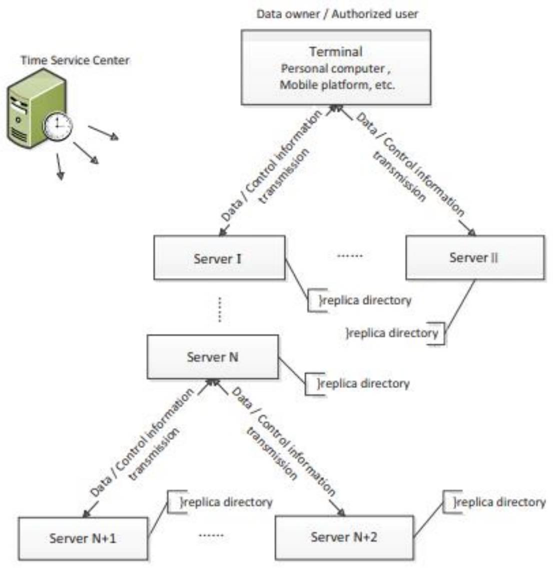


Figure 2.11: Replica associated model of MADS [9]

This model can delete replicated data based on two specific events, expired data storage period and cloud server receives delete instruction. One of the major downfalls of this model is that it relies upon the ethics and operations of the cloud storage service provider. It assumes that the cloud storage servers will follow the requested protocol operations.

Another model named Cryptographic Cloud Storage was proposed by Kamara et al. (2010) [10]. They have also considered the problem of making a cloud storage service secure on top of public cloud infrastructure where the service provider is not a trusted entity. The model consists of three components; a data processor (DP), a data verifier (DV) and a token generator (TG).

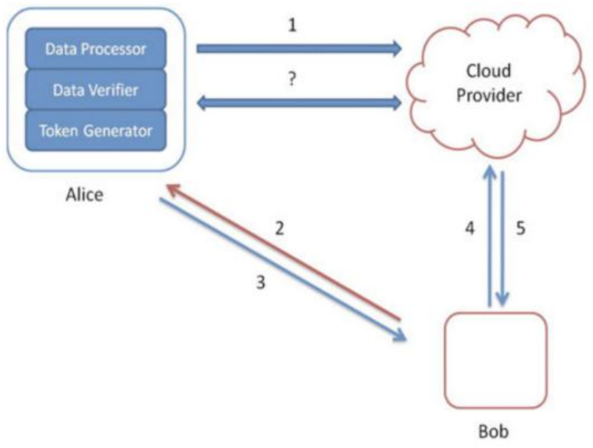


Figure 2.12: Consumer architecture [10]

DP processes data before sending to the cloud storage. DV checks if the data is tempered in the cloud and TG is used to generate tokens that allows the cloud service provider to retrieve customer data segments and a credential generator which implements access control policy by issuing credentials to different parties. Two different architectures are defined in this model, one for consumers and another for enterprises.

In the consumer architecture (shown in Figure 2.12), the data get prepared by Alice's DP before it is sent to the cloud. Bob then asks Alice for allowing him to look for a keyword. Alice's token and credential generators send token for the specified keyword and a credential to Bob. Bob then sends the token to the cloud and the token is used to find the corresponding encrypted documents. Then the documents are sent to Bob.

We will not discuss the enterprise architecture here. But both these architectures come with a minor vulnerability. Because, searchable encryption is used in both cases. In this encryption scheme, the index can be decrypted by providing a token for a keyword, which can be used to retrieve the pointers to the encrypted files. Now the problem with that is, some useful information could be leaked to the server. Because, over time, the server can make assumptions or guesses that some portion of the documents contain a word in common based on client's search pattern.

An IBE based design for assured deletion in cloud storage, proposed by Bentajer et al. (2019) [11] aims to build assured file deletion based on identity based cryptography. The primary components of this model are: a data owner, a cloud storage provider, and a key management system (KM) responsible for issuing users' private keys.

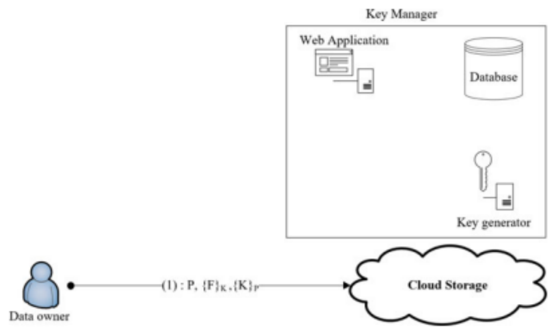


Figure 2.13: File upload in IBE based assured deletion in cloud storage [11]

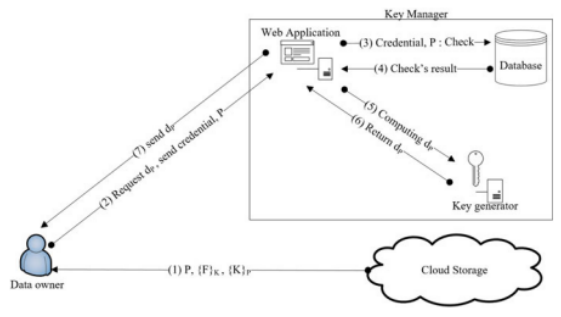


Figure 2.14: File download in IBE based assured deletion in cloud storage [11]

We will not dig deep into the details of this model. The file upload and download process is shown in Figure 2.13 and Figure 2.14. In this method, policy P renewal might be referred to as the policy revocation or deputation of decryption capabilities, as opposed to public key certificates which contain a preset expiration date. The public key ID || expiration_date is used so that the private corresponding key can be retrieved by the data owner until the expiration_date.

There are certain limitations of this model e.g. after the date has expired, the data owner needs to request a new private corresponding key that will be rejected by the Key Manager (KM) unless the expiration date policy is renewed. As this model mostly focuses on performance enhancement over FADE [5] design, the vulnerabilities of FADE [5] architecture might have been propagated into this model as well.

Another design was imposed by Sathe et al. (2018) [12] to address the issue of duplication in the cloud storage. They have designed an application based on block level based file deduplication scheme over fragmented blocks. They focused on guaranteed erasure using user defined policies. The keys are also deleted when a policy gets expired so that the file becomes inaccessible to anyone including the proprietor. One of the key limitations of this model is that, if any of the block/fragment of a file gets corrupted, the entire file becomes unusable.

A multi-layer trust security model (MLTSM) based on unified cloud platform trust was proposed by Sule et al. (2017) [13] that employs a fuzzy logic combination of on-demand states of different security mechanisms.

A novel block design-based key agreement protocol was proposed by Shen et al. (2018) [14]. It supports multiple participants. The number of participants can be extended flexibly according to the structure of the block design. This proposed model improved the security and efficiency of group data sharing in cloud computing through conference key agreement protocol. One downside of key agreement is that it asks for a large amount of information interaction in the system which increases the computational cost. To resolve this issue, they employed symmetric balanced incomplete block design SBIBD in their protocol design.

There are many other models proposed to ensure secure cloud environment e.g. in 2013 an Attribute Based Encryption (ABE) based secure document self-destruction (ADS) scheme was proposed by Xiong et al. (2013) [15]. This scheme works by making documents dispose automatically after a certain amount of time (specified by user). Another model proposed by Xiong et al. (2014) [16] which is an ABE based system which uses key-policy with time-specified attributes (KP-TSABE). In this model, decryption mechanism works only if the time instance is within the acceptable time interval and the key's access structure is satisfied by the associated attributes of the cipher. In the same year, another self destructive model named SelfDoc was proposed by Xiong et al. (2014) [17] which allows self destruction of composite documents. All these systems have their own limitations and complexities.

We also came across some works that may not be directly related to our project but surely had an impact on our work. One such work was a private data disclosure checking method proposed by Changbo et al. (2017) [18] that can be applied to the collaboration interaction process among SaaS Services which can help prevent users' private data from being used and propagated illegally.

As we wanted to ensure that our model does not rely upon any trusted third party, one interesting work which helped us brainstorm was "A "No Data Center" Solution to Cloud Computing" by Mengistu et al. (2017) [19]. In this paper, they proposed an opportunistic Cloud Computing system which they called cuCloud, that runs on scavenged resources of underutilized PCs within an organization/community. The idea is to reduce the cost of maintaining data centers and large server machines and utilizing the computing devices that are not being fully utilized by individuals or organizations. In spite of having limitations such as insufficient security measures, dynamic and unreliable nature of the member hosts; it paved us a way to design our untrusted server.

In the next chapter, we will disclose the design of our proposed model. We will also discuss how it works and how it solves the issues arose with previously proposed models. We will also talk about why we think it is more secure than any of the previously proposed systems.

3 Design of the Proposed Model

Our model consists of two distinct components; a client application and a server. Client side application will handle all the complex cryptographic operations as well as network communication with both server and cloud storage service. On the other hand, server is connected with a database which is used to store and retrieve information. At first, user needs to register by providing his/her email, passphrase and recovery information. Then he/she needs to sign in using the same credentials that were used during registration. After signing in, the user can perform operations such as upload, download, share and delete files from cloud storage. All these operations are cryptographically secure.

When a user chooses to upload a file, that file is first encrypted by the client application on the user's end using a randomly generated key. The key is then encrypted by the user's passphrase. After the file encryption is complete, the file is uploaded to the cloud. If cloud service API provides functionality to upload the file as a stream of bytes and the encryption algorithm supports partial encryption, it is possible to encrypt and upload the file simultaneously. So, the file stored in cloud storage can only be accessed by the user who has uploaded the file.

If a user wants to download a file, that file is first downloaded by the client application. After the download is complete, the file remains unusable as it is the encrypted version of the file. Then the client application retrieves the encrypted version of random key from server side. That key is then decrypted using the user's passphrase. The downloaded file is then decrypted using the random key. Similar to upload operation, it is possible to download and decrypt the file simultaneously depending on the cloud service API and the decryption algorithm.

When a file is shared with another user, the second user can access the file. Suppose Alex wants to share a file with Bob. The client application will retrieve the encrypted version of the random key that was used to encrypt the file. Then the key will be decrypted by Alex's passphrase and re-encrypted by Bob's public key. Bob can then download the shared file using his private key. This way, only Alex and Bob can access the file. Alex can also revoke Bob's access.

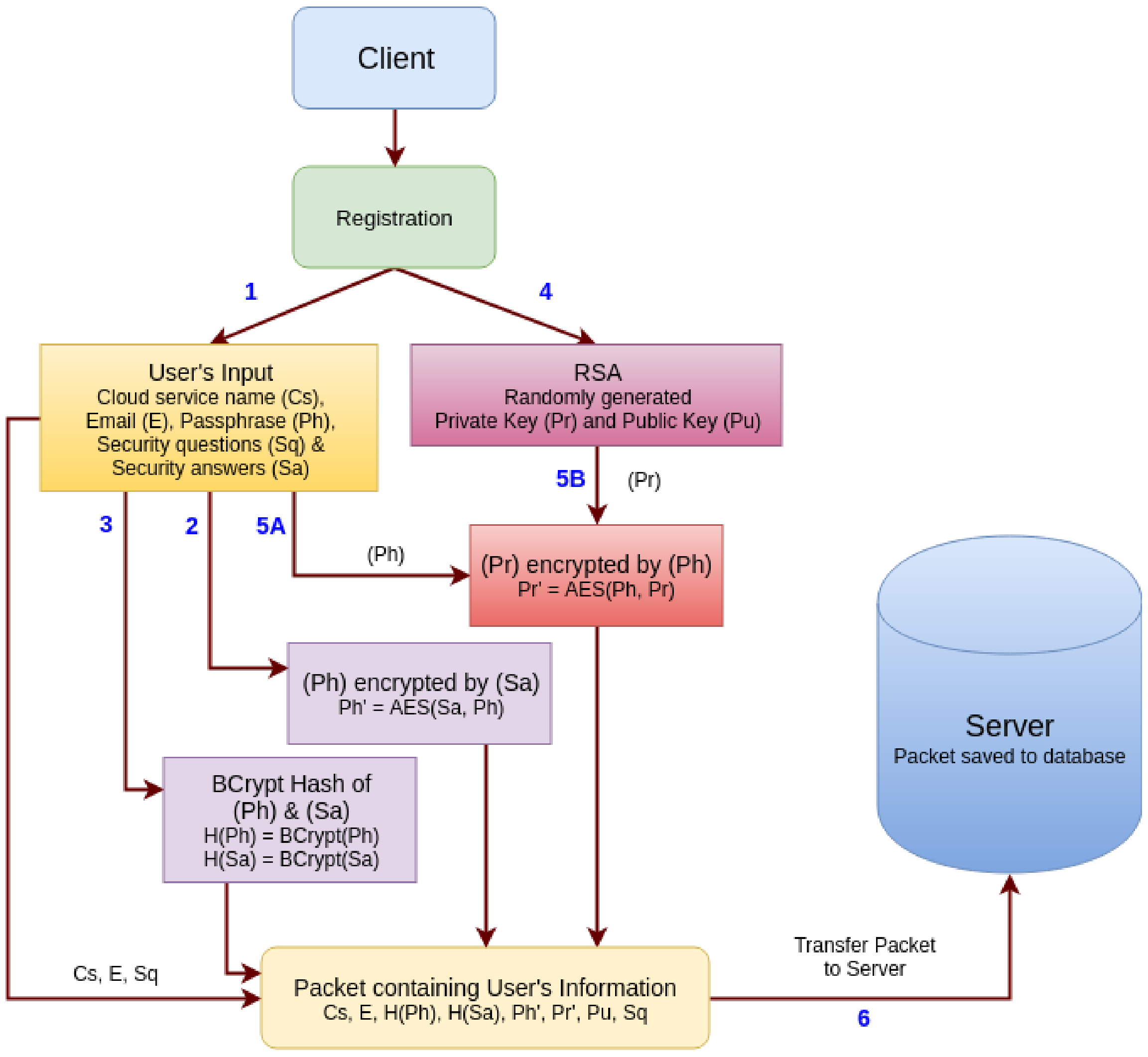
If a file is deleted by a user, that file becomes inaccessible. Even the shared files can no longer be accessible by the user with whom the files were shared. From the previous example, if Alex deletes the file from cloud storage, Bob can no longer access the file. As the files kept in cloud storage are encrypted, replicated copies of the files kept by cloud service provider become obsolete. Thus assured file deletion is achieved.

Our model also supports forgotten passphrase recovery. If it so happens that a user forgets his/her passphrase, he/she can recover the passphrase by providing the same recovery information that was used during registration. An encrypted version of passphrase is kept in the server. When a user requests passphrase recovery, the client application fetches the encrypted passphrase and decrypts that using the recovery information provided by the user. User must enter a new passphrase to complete the recovery process.

In the above discussion, we have tried to reduce the technical complexity as much as possible to make sure that anyone having any amount of knowledge regarding cryptography can understand the basic architecture of our proposed model. We have an in-depth discussion below on each operation individually which demonstrates how each operation works, why our proposed model is more secure than any of the models proposed previously and most importantly, why the server side does not require to be a trusted entity.

3.1 User Registration

User registration is the most vital element of our proposed model. During the registration process, user needs to provide some information. Client application then performs some cryptographic operations to the sensitive portion of user information before sending them to the server.



Client
Registration
1
4
User's Input
Cloud service name (Cs), Email (E), Passphrase (Ph), Security questions (Sq) & Security answers (Sa)
RSA
Randomly generated
Private Key (Pr) and Public Key (Pu)
5B
(Pr)
3
2
5A
(Ph)
(Pr) encrypted by (Ph)
Pr' = AES(Ph, Pr)
(Ph) encrypted by (Sa)
Ph' = AES(Sa, Ph)
Server
Packet saved to database
BCrypt Hash of (Ph) & (Sa)
H(Ph) = BCrypt(Ph)
H(Sa) = BCrypt(Sa)
Cs, E, Sq
Packet containing User's Information
Cs, E, H(Ph), H(Sa), Ph', Pr', Pu, Sq
Transfer Packet to Server
6

Figure 3.1: User registration process flow diagram

The entire registration process is described below:

1. At first, user needs to provide his/her email address, passphrase, cloud service name, and recovery information (optional).
   a. ***Email address:*** It is required to uniquely identify the user.
   b. ***Passphrase:*** It is used for signing in and required by both encryption and decryption processes. It must be at least 8 characters long.

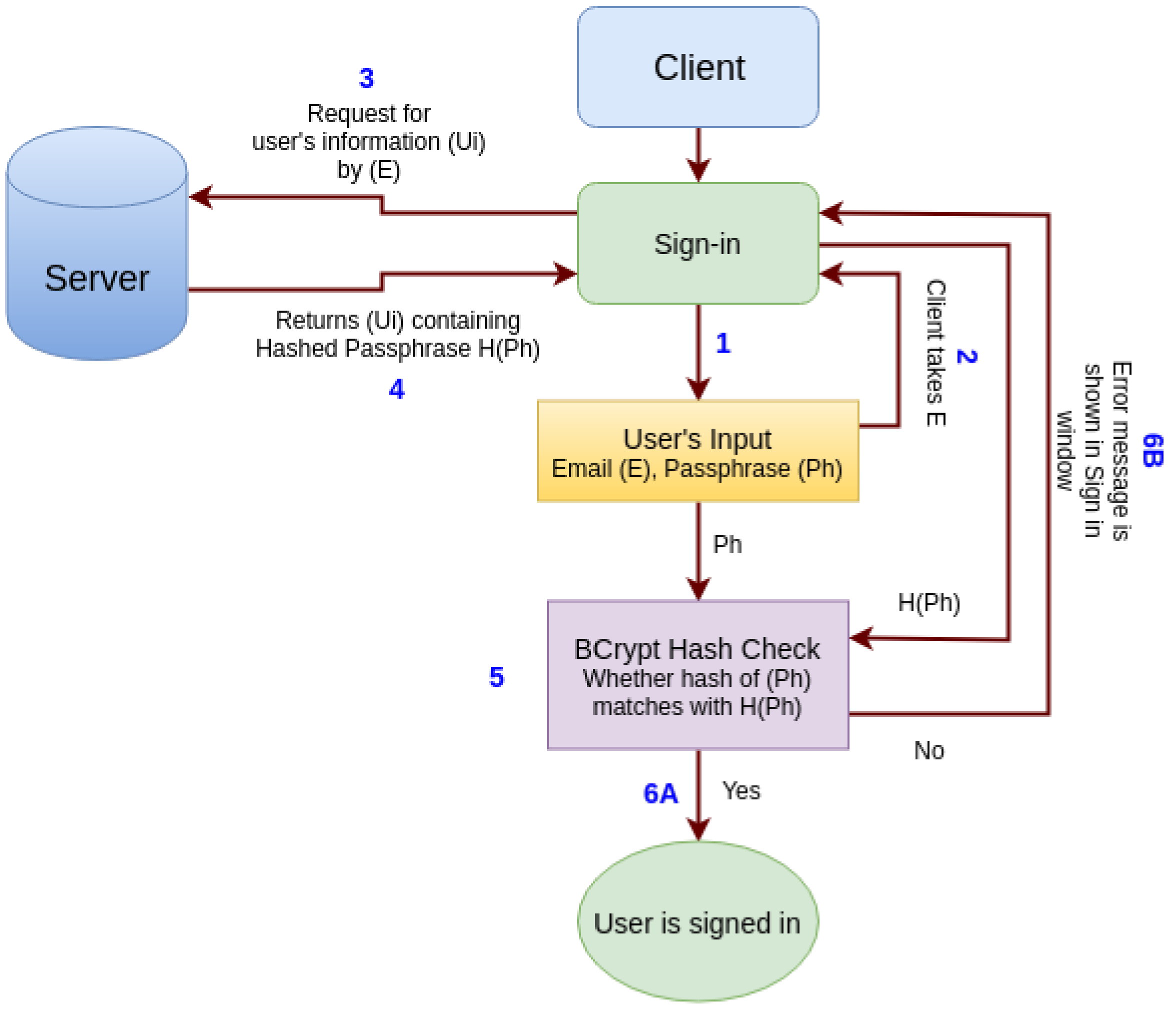
   c. ***Cloud service name:*** Cloud service name (e.g. Google Drive, Dropbox etc.) is used to distinguish between user accounts when same email address is associated with different cloud storage service. e.g. A user may have a dropbox account and Google Drive account associated with the same email.
   d. ***Recovery information (optional):*** Recovery information consists of Security questions and the corresponding answers. If a user forgets his/her passphrase, he/she can recover the account by answering the security questions. Users may put any security question they prefer. Even though providing security question information is optional, it is highly recommended.
2. If recovery information (security questions, Sq and answers, Sa) are provided by the user, client application encrypts user's passphrase (Ph) by the security answers (Sa) using AES. e.g. A user has provided multiple security questions. Now passphrase will be encrypted by the first answer. The encrypted passphrase will be re-encrypted by the next answer and so on.
3. Hash of the user provided passphrase, H(Ph) and Security answers, H(Sa) are generated using BCrypt hash algorithm as it can adapt to hardware improvements and remain secure in the future as well [20]. Hash of the passphrase, H(Ph) is required during sign-in to check if the user has provided the correct passphrase. And hashed security answers, H(Sa) are required during forgotten passphrase recovery process to check if the security questions (Sq) have been answered correctly.
4. After that, a pair of random RSA public (Pu) and private key (Pr) is generated. RSA keys are required for sharing a file securely. Public key (Pu) is used for encryption and the private key (Pr) is used for decryption. We have an extensive discussion regarding file sharing process later in this chapter.
5. The private key (Pr) is then encrypted by the user provided passphrase using AES. As RSA key generation is computationally intensive, we do not want to generate private key (Pr) every time user wants to upload or download a file. So, this step is necessary in order to avoid redundant RSA key generation.
6. Lastly, cloud service name (Cs), email (E), hashed passphrase H(Ph), public key (Pu), encrypted private key (Pr'), security questions (Sq), hashed security answers H(Sa), encrypted passphrase (Ph') are sent to the server by the client application. Sq, H(Sa) and Ph' are only sent to the server if user provides recovery information.

The complete registration process is described using a flow diagram in Figure 3.1.

Now a question might arise that, with all these information stored in the server, what will happen if the server side is compromised? Or, the database gets altered by the administrator or any intruder? As we have mentioned earlier, we designed the server side in such a way that it does not require to be a trusted entity. Passphrase (Ph) is kept encrypted by security answers (Sa) and only the hash of the security answers, H(Sa) is kept in the server. Private key (Pr) is also encrypted by passphrase (Ph). So, even if the server is compromised, attackers cannot get any useful information out of it. Even if the database gets altered, user's files will remain unrecoverable. But in this case, the user will also lose access to his/her files.

3.2 User Sign-in

We decided to have sign-in functionality so that users will not have to type in the passphrase again and again for different operations like upload, download or even sharing files. After a user has been registered in our service, he/she must sign-in.



Client
3
Request for user's information (Ui) by (E)
Server
Sign-in
Returns (Ui) containing Hashed Passphrase H(Ph)
4
1
2
Client takes E
6B
Error message is shown in Sign in window
User's Input
Email (E), Passphrase (Ph)
Ph
H(Ph)
5
BCrypt Hash Check
Whether hash of (Ph) matches with H(Ph)
No
6A
Yes
User is signed in

Figure 3.2: User sign-in process flow diagram

The sign in process consists of the following steps:

1. At first, user needs to provide his/her credentials (Email, E and Passphrase, Ph) to the client application.
2. The client application receives the user's input and takes the email address (E) into account.
3. Client application then requests the server for user information (Ui) corresponding to the provided email address.
4. Server returns the appropriate user information (Ui) associated with the email address (E). User information (Ui) contains previously stored hash value of the user's passphrase, H(Ph).
5. Client application then checks whether the previously stored hash of the user's passphrase H(Ph) matches the passphrase provided in step 1.
6. If both the hash values match, the user is signed in successfully and he/she will be able to access all the functionalities of the client application. But if the hash values do

not match, the user will not be allowed to use functionalities such as; upload, download, share etc.

It is worth mentioning that, if the server database is altered by any intruder, the intruder might be able to sign in using the user's account but will not be able to access any of the files. But the user will also lose access to his/her files and may not be able to sign in. The complete flow of the sign-in process is described using a diagram in Figure 3.2.

3.3 Upload File to Cloud Storage

The upload operation is pretty straight forward in our model. We just added a layer of encryption before uploading the file to the server. This will ensure that nobody but the uploader can access the file.

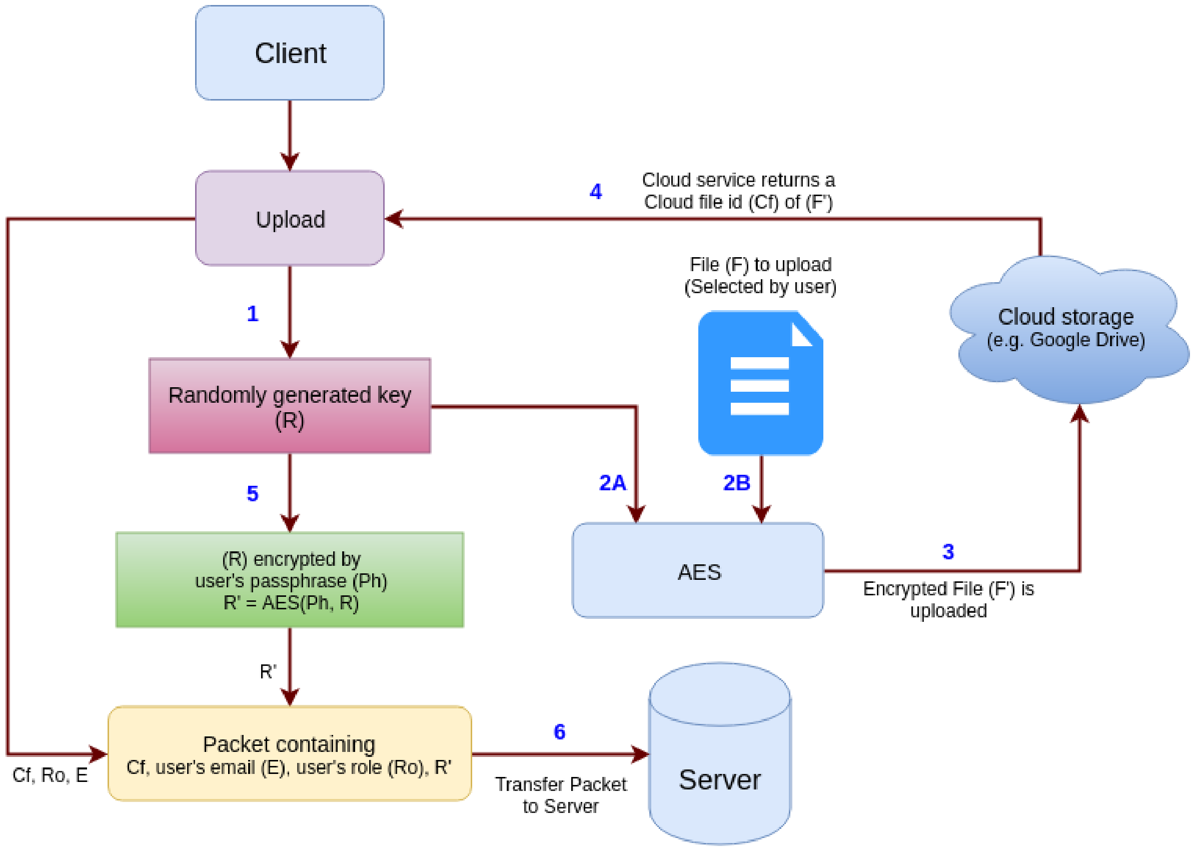

Client
Upload
4
Cloud service returns a Cloud file id (Cf) of (F')
File (F) to upload (Selected by user)
Cloud storage (e.g. Google Drive)
1
Randomly generated key (R)
5
2A
2B
AES
3
Encrypted File (F') is uploaded
(R) encrypted by user's passphrase (Ph) R' = AES(Ph, R)
R'
Cf, Ro, E
Packet containing Cf, user's email (E), user's role (Ro), R'
6
Transfer Packet to Server
Server

Figure 3.3: File upload process flow diagram

The upload operation is described below:

1. When user requests for uploading a file, the client application generates a random key (R). The random key is way too strong to be guessed. It is a variable length key. Which means, the length of the key is not fixed. It could be of any length between 64 to 128 bytes. It can contain both uppercase and lowercase characters, digits and special characters e.g. exclamation mark (!), asterisk (*) etc.
2. The random key (R) is used to encrypt the selected file (F). To achieve stronger encryption, we have decided to use 256 bit variant of AES algorithm.
3. The encrypted file (F') is uploaded to the cloud storage.

4. After receiving the encrypted file (F') successfully, cloud service provider will send a unique id (Cf) for that file as response.
5. The random key (R) that we generated in step 1, is encrypted by the user's passphrase (Ph). As upload operation can only be performed by signed in user, the passphrase (Ph) is already stored in the session of the client application. So, user will not have to enter his/her passphrase again.
6. Lastly, unique id provided by cloud service (Cf), user's email (E), and the encrypted random key (R') along with one small piece of information set by the client application, user's role (Ro) are sent to the server. Thus the upload process is completed.

User's role, Ro is used to determine if the user is the owner of the file or the file has been shared. The use case of this tiny information is described later in this chapter. Entire file upload process is described using a flow diagram in Figure 3.3.

3.4 Share Files with Others

When we upload a file to the cloud, it is often necessary to share that with our friends or colleagues. Initially it might seem a bit odd that, why would we even need to share a file that we want to keep secret? Suppose a user, who is a businessman, needs to keep a document (e.g. contract, agreements etc.) private between himself and his/her business partner. In such a scenario, the easiest way to share a secret file is by sharing the passphrase with partner. Isn't it? Well, the answer is yes. The business partner will be able to retrieve that file by decryption using the passphrase. But there is one big problem. The business partner will also get access to any other secret file that the user (businessman) has which he wanted to be accessible by himself. Now one workaround could be using a different passphrase for the file to be shared with the business partner. Although this solves the issue of privacy but it creates another issue of remembering lots of passphrases. e.g. a businessman has multiple business agreements and he wants to share each agreement with individual business partners. In that case if he has 10 business partners to share 10 different files, he will have to remember 10 individual passphrases which would be a mess.

In SFADE+ [7] usage of RSA encryption/decryption was proposed for sharing files. It requires a repository for keeping public key of users. The private key is generated based on user's passphrase. This method works flawlessly to share a file secretly with another user except for one drawback.

RSA cryptosystem is well known for heavy resource consumption specially key pair generation and decryption. In SFADE+ [7] it is required to generate private key every time user wants to share a file. In our proposed model, we have minimized the number of times a pair of RSA keys is generated, keeping the privacy intact. In section 3.1, where we discussed about user registration process, we also talked about RSA key generation. That is the only time in our proposed system when RSA key is generated.

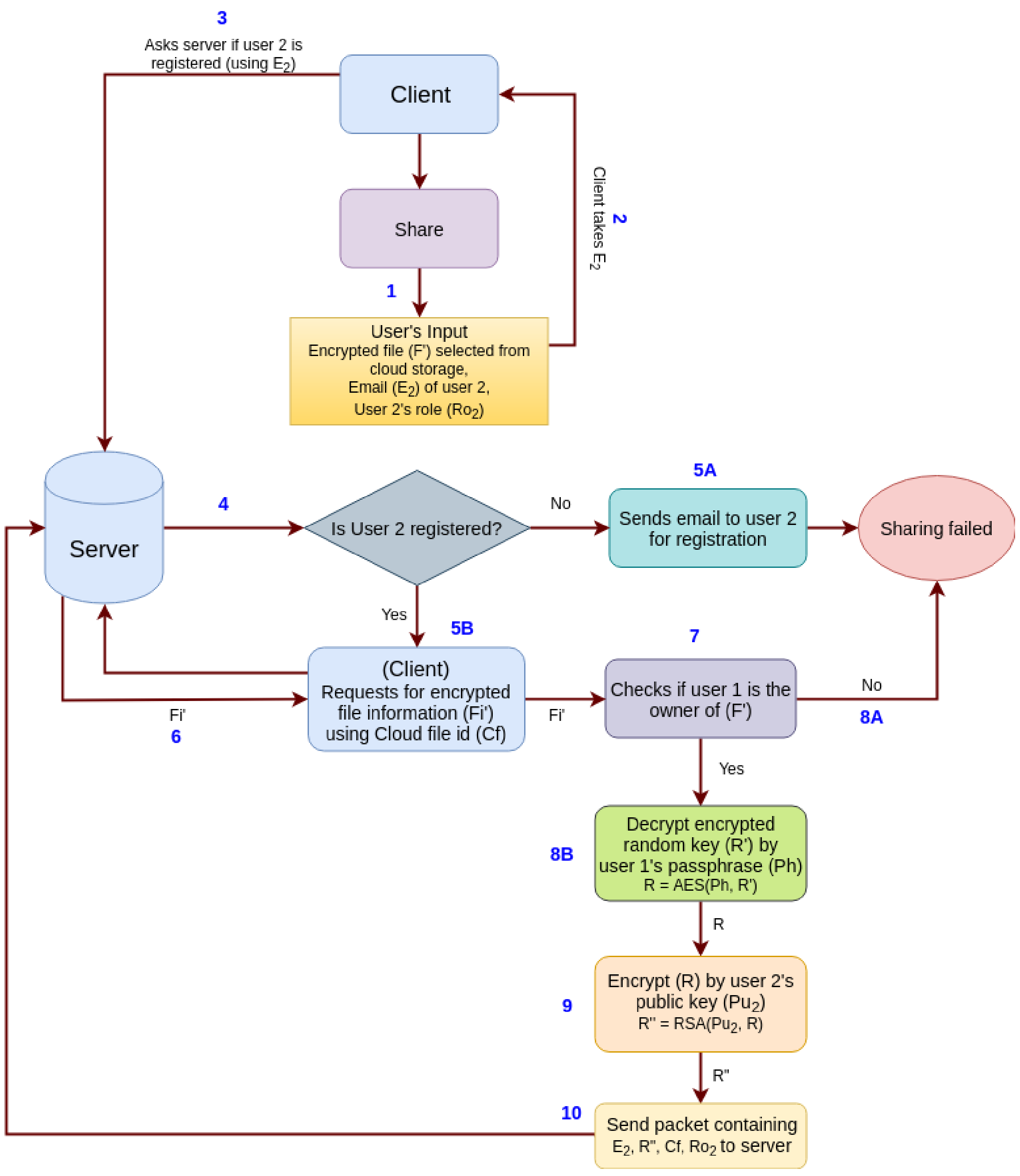

3
Asks server if user 2 is registered (using E_2)
Client
Share
1
2
Client takes E_2
User's Input
Encrypted file (F') selected from cloud storage,
Email (E_2) of user 2,
User 2's role (Ro_2)
Server
4
Is User 2 registered?
No
5A
Sends email to user 2 for registration
Sharing failed
Yes
5B
(Client)
Requests for encrypted file information (Fi') using Cloud file id (Cf)
Fi'
6
Fi'
7
Checks if user 1 is the owner of (F')
No
8A
Yes
8B
Decrypt encrypted random key (R') by user 1's passphrase (Ph)
R = AES(Ph, R')
R
9
Encrypt (R) by user 2's public key (Pu_2)
R'' = RSA(Pu_2, R)
R''
10
Send packet containing E_2, R'', Cf, Ro_2 to server

Figure 3.4: File sharing process flow diagram

Below we will discuss the entire file sharing process flow step by step:

1. At the beginning of the file sharing process, user needs to provide the email address (E_2) and access role (Ro_2) of user 2 with whom the file needs to be shared.
2. The client application receives the user's input and takes user 2's email address (E_2) into account.
3. The client application will communicate with server to check if the provided email address (E_2) is associated with any account.
4. Server will check if user 2 is registered.
5. If no user account is associated with (E_2), an email will be sent to the email address asking to register in our service to receive the file. But if user 2 is registered in our

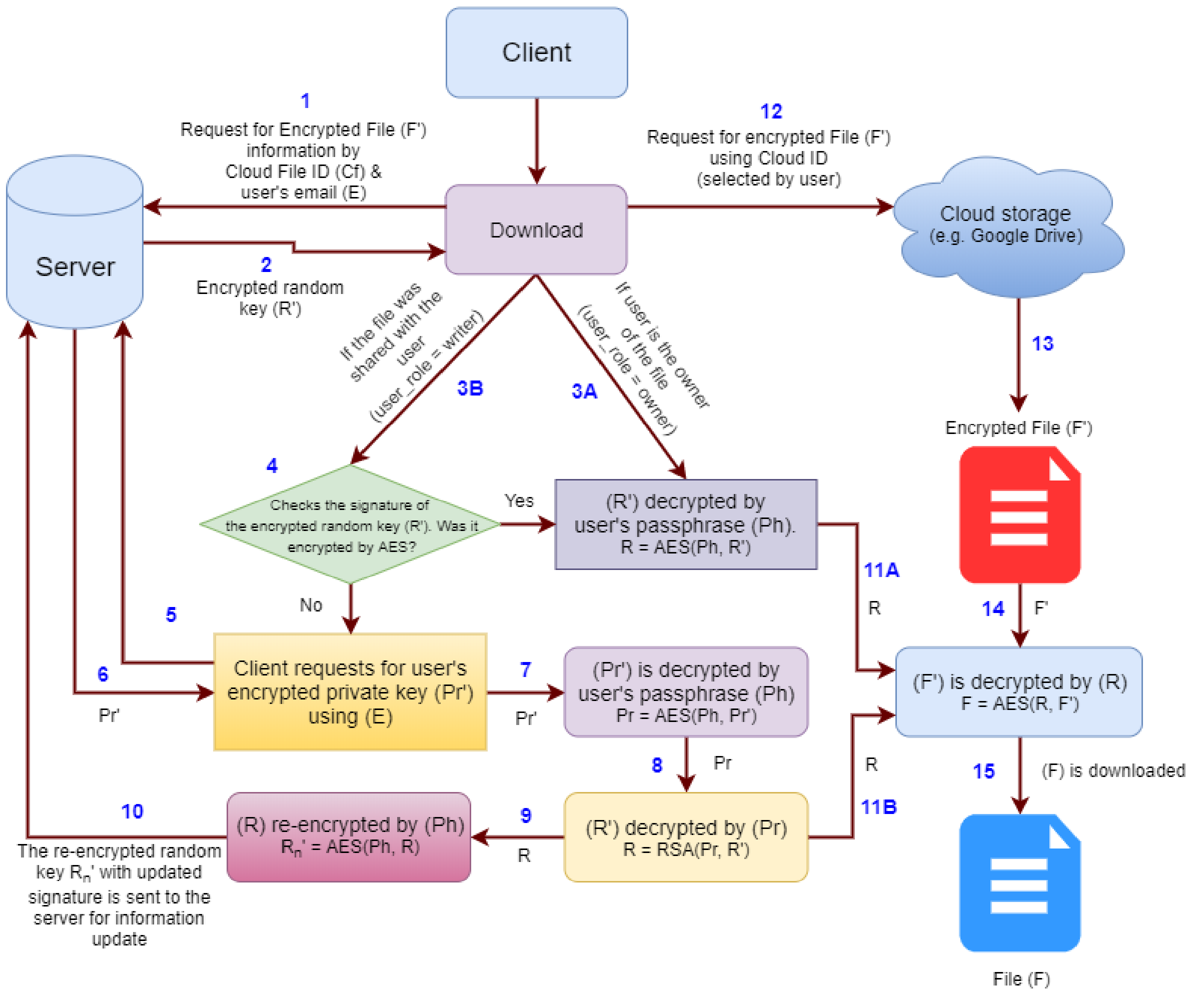
service, client application will request server for encrypted file information (Fi') using the cloud file id (Cf).

6. Server will provide the encrypted file information (Fi') to the client application.
7. The client application will then check if user 1 is the owner of the file (F').
8. If user 1 is not the owner, sharing will fail. Otherwise, client application will decrypt the encrypted random key (R') using user 1's passphrase (Ph) to retrieve plain random key (R).
9. Plain random key (R) will then be re-encrypted by user 2's public key (Pu_2).
10. In the end, a packet is formed combining email address of user 2 (E_2), re-encrypted random key (R''), cloud file id (Cf) and user 2's access role (Ro_2) and sent to the server.

Server stores this packet in its database so that it can be used by user 2's client application while downloading the file (F').

3.5 Download File from Cloud Storage

Download section is usually placed under upload section. Because without being able to download files, uploading and sharing makes no sense. But placing file sharing section before download conveys a great significance.


Client
1
Request for Encrypted File (F') information by Cloud File ID (Cf) & user's email (E)
12
Request for encrypted File (F') using Cloud ID (selected by user)
Server
Download
Cloud storage (e.g. Google Drive)
2
Encrypted random key (R')
If the file was shared with the user (user_role = writer)
If user is the owner of the file (user_role = owner)
3B
3A
13
Encrypted File (F')
4
Checks the signature of the encrypted random key (R'). Was it encrypted by AES?
Yes
(R') decrypted by user's passphrase (Ph). R = AES(Ph, R')
11A
R
14
F'
No
5
6
Pr'
Client requests for user's encrypted private key (Pr') using (E)
7
Pr'
(Pr') is decrypted by user's passphrase (Ph) Pr = AES(Ph, Pr')
(F') is decrypted by (R) F = AES(R, F')
8
Pr
R
11B
15
(F) is downloaded
10
The re-encrypted random key R_n' with updated signature is sent to the server for information update
(R) re-encrypted by (Ph) R_n' = AES(Ph, R)
9
R
(R') decrypted by (Pr) R = RSA(Pr, R')
File (F)
Figure 3.5: File download process flow diagram

We have designed our download process in a way that it can download the files shared to a user along with the files that were uploaded by him/her. User will not even notice the difference while downloading a file regardless of the file being shared or uploaded by him/her.

File download operation is performed in the following steps:

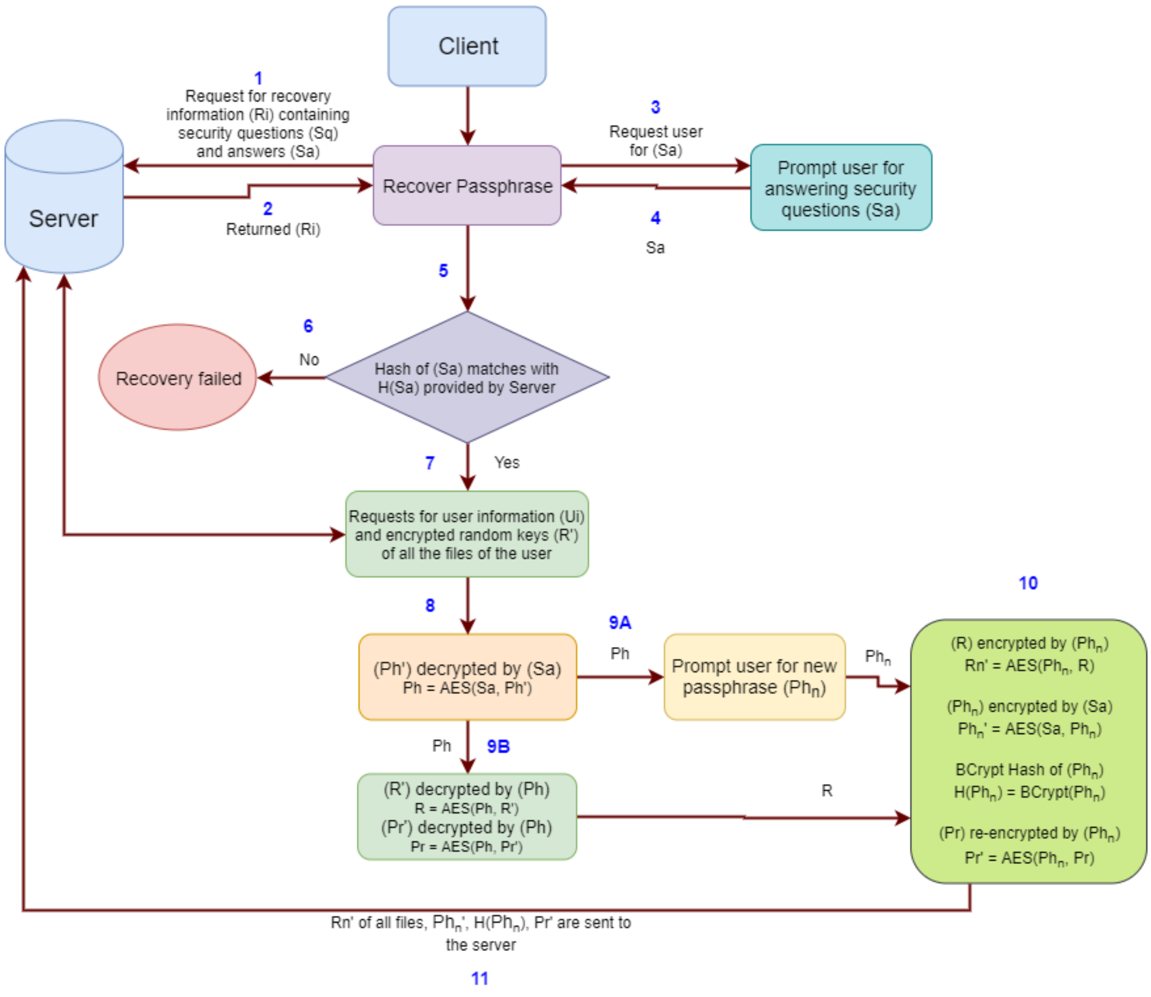
1. When a user requests for downloading a file, the client application will request the server for information regarding the encrypted file (F').
2. If the user requesting for file does not have permission to access the file, the server will return an error message. Otherwise, server will send the file information (of the encrypted file) to the client application which contains encrypted random key (R').
3. After receiving the information, client will check the access role of the user.
4. If the user is the owner of the file *OR* the user's role is "writer" and the signature of the encrypted random key (R') indicates that it was encrypted by AES, R' is decrypted by the user's passphrase using AES. It is a five byte signature which is used to determine whether the random key was encrypted using user's passphrase using AES or it was encrypted using the user's private key using RSA.
5. Now, if the user's role is "writer" and the signature of the encrypted random key (R') indicates that it was encrypted by RSA, the client application will request the server to provide the user's encrypted private key (Pr').
6. Server then sends the user's encrypted private key (Pr') to the client application.
7. Encrypted private key (Pr') is then decrypted by user's passphrase using AES to retrieve the private key (Pr).
8. Then the encrypted random key (R') is decrypted by the private key (Pr) using RSA.
9. The random key (R) is now found which can be used to decrypt the file from cloud storage. But before requesting cloud storage for the file and decrypting it, the client application encrypts the random key (R) by user's passphrase using AES and the signature is added which resembles that the newly encrypted random key (R_n') is encrypted using the user's passphrase.
10. The newly encrypted random key (R_n') is then sent to the server and the file information is updated. The reason behind this step is described later in this section.
11. At this stage, whether the user is the owner of the file or the file was shared with the user, the client application has retrieved the random key (R).
12. The client application then requests the cloud storage service for the encrypted file (F').
13. The cloud storage service will provide the encrypted file to the client application.
14. After receiving the file, client application will decrypt the file using the random key (R).
15. Thus, the file download process completes and the file ready to use.

Now, the additional encryption performed in step 9 might seem redundant at first glance but this step was kept in our design on purpose. In section 3.4, where we described our file sharing process, we mentioned that RSA decryption is computationally intensive. So, we did not want to use RSA decryption every time a shared file is downloaded by the user. That is why when a shared file is downloaded for the first time by a user (with whom the file was shared), we are simply re-encrypting the random key (R). So in total, our system runs RSA decryption only once for each shared file which makes our model less resource consuming.

3.6 Recover Forgotten Passphrase

After all these security precautions with encryption/decryption process, everything will be shattered if the passphrase is forgotten. As we have already mentioned, none of the existing models provide any solution to this problem. So, if the passphrase is forgotten, all the secure data become obsolete. There is no way to recover the actual content from those data.
In our proposed model, we focused on this issue and designed a system to help users recover their lost passphrase. In the first section of this chapter (section 3.1), where we discussed user registration, there was a user provided information for recovery purpose. We kept this piece of information optional to make the registration process less time consuming for those who want to get instant access to our service.

The passphrase recovery depends on some security questions and their answers. Both questions and answers are set by the user during registration. For using passphrase recovery, user will be provided the questions that were set by him/her and he/she must provide the exact same answer. If the answers match, the user will be asked to enter new passphrase. A point worth noting that the answers are not case sensitive. After the recovery process is complete, the user can sign in using the new passphrase. The files encrypted by old passphrase will also be accessible without any issue.



Client
1
Request for recovery information (Ri) containing security questions (Sq) and answers (Sa)
Server
2
Returned (Ri)
Recover Passphrase
3
Request user for (Sa)
Prompt user for answering security questions (Sa)
4
Sa
5
6
No
Recovery failed
Hash of (Sa) matches with H(Sa) provided by Server
7
Yes
Requests for user information (Ui) and encrypted random keys (R') of all the files of the user
8
(Ph') decrypted by (Sa)
Ph = AES(Sa, Ph')
9A
Ph
Prompt user for new passphrase (Phn)
Phn
10
(R) encrypted by (Phn)
Rn' = AES(Phn, R)
(Phn) encrypted by (Sa)
Phn' = AES(Sa, Phn)
BCrypt Hash of (Phn)
H(Phn) = BCrypt(Phn)
(Pr) re-encrypted by (Phn)
Pr' = AES(Phn, Pr)
Ph
9B
(R') decrypted by (Ph)
R = AES(Ph, R')
(Pr') decrypted by (Ph)
Pr = AES(Ph, Pr')
R
Rn' of all files, Phn', H(Phn), Pr' are sent to the server
11

Figure 3.6: Passphrase recovery process flow diagram

Each step of passphrase recovery process is described below:

1. At the beginning of the passphrase recovery process, the client application requests server for recovery information (Ri) of the user.
2. The server accepts the request and returns the appropriate recovery information (Ri) of the user. It is important to note that recovery information (Ri) contains security questions, Sq and hashed security answers, H(Sa).
3. The client application then asks the user to answer those questions.
4. Client application receives the security answers (Sa) from the user’s input.
5. Then it checks whether the hash of the user provided answers match the hashed answers H(Sa) provided by the server.
6. If the hash values do not match, the recovery process fails immediately.
7. But if the values match, the client application retrieves user information (Ui) and encrypted random key (R’) of all the files of that user from the server. It is worth mentioning that, user information (Ui) contains information like encrypted passphrase (Ph’).
8. Then it decrypts the encrypted passphrase (Ph’) by the answers (Sa) using AES and thus the old passphrase is retrieved.
9. The user is then asked to enter a new passphrase. And all the random keys (R’) of user’s files and the user’s private key (Pr) will be decrypted by old passphrase (Ph) using AES.
10. All the random keys (R’) of user’s files and the user’s private key (Pr) will be re-encrypted by new passphrase (Ph_n). The new passphrase is encrypted by the answers (Sa). It is worth mentioning that in all these cases, AES is used. And lastly, BCrypt hash of the new passphrase $H(P_{hn})$ is generated.
11. The newly encrypted random keys (Rn’), encrypted new passphrase (Ph_n’), hash of the new passphrase $H(Ph_n)$ and encrypted private key (Pr’) are packed in and sent to the server to update the existing information.

And in this way our passphrase recovery process completes. It is a bit lengthy process in terms of the number of operations needed to be performed in order to recover old information. But we assumed that this function will not be run too frequently compared to other functions (e.g. sign up, sign in etc.)

4 Implementation Details

Without practical implementation, theoretical models do not convey much value to the real world. So we wanted to have our proposed model implemented in a way that it can be used in day-to-day life. During implementation, our main focus was to make it as simple as possible so that anyone with very little technical knowledge can operate the software. And at the same time, we did not want to compromise user's privacy/security in the cloud. So we are calling our implementation "CryptedCloud".

In this chapter, we will be focusing on the implementation details of our proposed model. We will describe how our model is implemented, which technologies we have used and why, which frameworks we used, which database we used along with schema diagram and also the design principles we followed during our implementation. As we have already mentioned in Chapter 3, that our model comprises two distinct components; a client application and a server side application. We will have separate discussion regarding the client application and the server application.

4.1 Client Application

We are calling the client side application of our implementation "CryptedCloud Client". We decided to make our client application cross-platform. Initially it was planned to build this as a web application. It would help us to run the same application across all the devices regardless of hardware capabilities or operating systems they are running (e.g. PC, Mobile devices etc.). But that would cause serious privacy leak as web applications are basically run on servers. Which means, all the sensitive information (e.g. passphrases, random keys etc.) will be sent to the server in plaintext. But it requires the server to be a trusted entity which violates our proposed model's purpose.

4.1.1 Technologies Used

So we needed to go for an alternate solution which is also cross platform and can be easily ported to mobile devices (at least those which are running Android operating system). So the most obvious choice for us was to write the entire client side in Java 8. The user interface (UI) was designed in Java Swing and a small portion of it uses JavaFX. This would allow us to run the application on the user's device and we could also optimize the performance of the application.

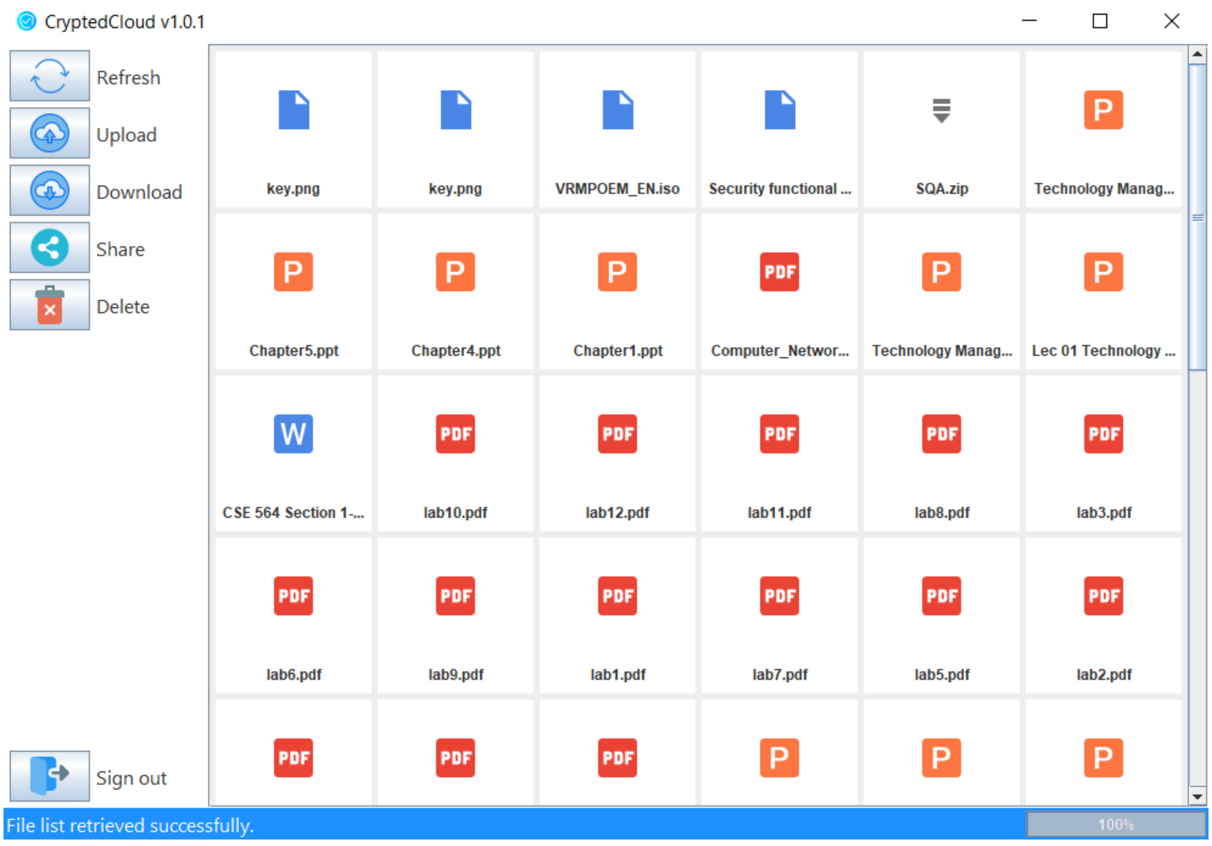
Then we had to decide which cloud service provider we will be working on. There were several options e.g. Google Drive, Dropbox, Amazon AWS etc. We chose to go with Google Drive as it is one of the free solutions available. For interacting with Google Drive from our client application, we used Google Drive API v3 [21]. For Google Drive authentication, we have used OAuth v2.0 [22].

Our client server communication is done over JSON (JavaScript Object Notation). So for ease of use, we have decided to use "Jackson" [23] library to help us parse the JSON data. It can convert JSON string to Java objects and vice versa.

4.1.2 Design Principles

During the design phase of our CryptedCloud Client, we focused on both external and internal design. We followed some design principles that would make our client application more modular with zero compromise to the performance.

The exterior or the user interface (UI) was designed with user friendliness in mind. Most of the complex operations are performed in the background without the requirement of user interaction.



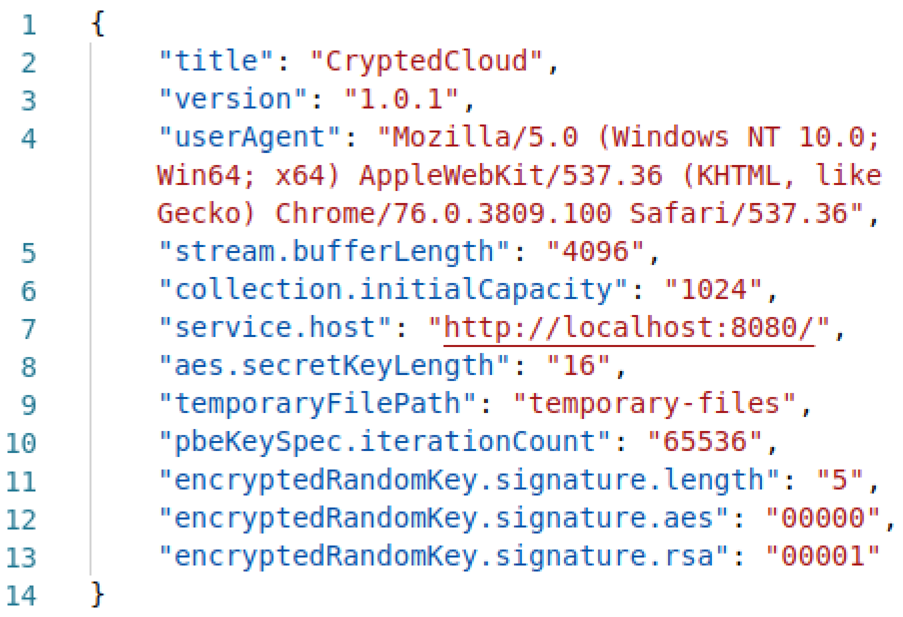
File list retrieved successfully.

Figure 4.1: Basic user interface of CryptedCloud Client

The basic user interface of the CryptedCloud Client is shown in Figure 4.1. Only a single portion of the UI is shown in this figure. We will cover the rest of the parts in Section 4.1.3, Functional specification later in this chapter which will have the details of each operation that can be performed through the UI.

The internal architecture is designed in a very modular manner. There are two separate threads working concurrently. One is the main thread or in our case it is the UI thread. The other is the background thread. UI thread handles all the user interactions that take place in the UI. On the other hand, background thread performs all the operations e.g. communicating to the server, retrieving data from cloud storage, performing all the cryptographic operations like encryption/decryption, hashing, etc. These two threads are separated for users to have a lag free experience. Otherwise, whenever a time consuming operation is performed, UI would remain stuck until that operation is finished. e.g. user is downloading a file of size 1 GB. Downloading and decrypting such a big file would take a noticeable amount of time. If it were a single threaded application, it would have been stuck until 1 GB is completely downloaded and decrypted.

To make this application more modular, we are specifically taking two measures. The first one is to read constant values from a configuration file. It will allow us to change the values anytime without having to change the source code. The configuration file is written in JSON format. As JSON data is easier to read, the configuration can be modified with ease.



```json
{
    "title": "CryptedCloud",
    "version": "1.0.1",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36",
    "stream.bufferLength": "4096",
    "collection.initialCapacity": "1024",
    "service.host": "http://localhost:8080/",
    "aes.secretKeyLength": "16",
    "temporaryFilePath": "temporary-files",
    "pbeKeySpec.iterationCount": "65536",
    "encryptedRandomKey.signature.length": "5",
    "encryptedRandomKey.signature.aes": "00000",
    "encryptedRandomKey.signature.rsa": "00001"
}
```

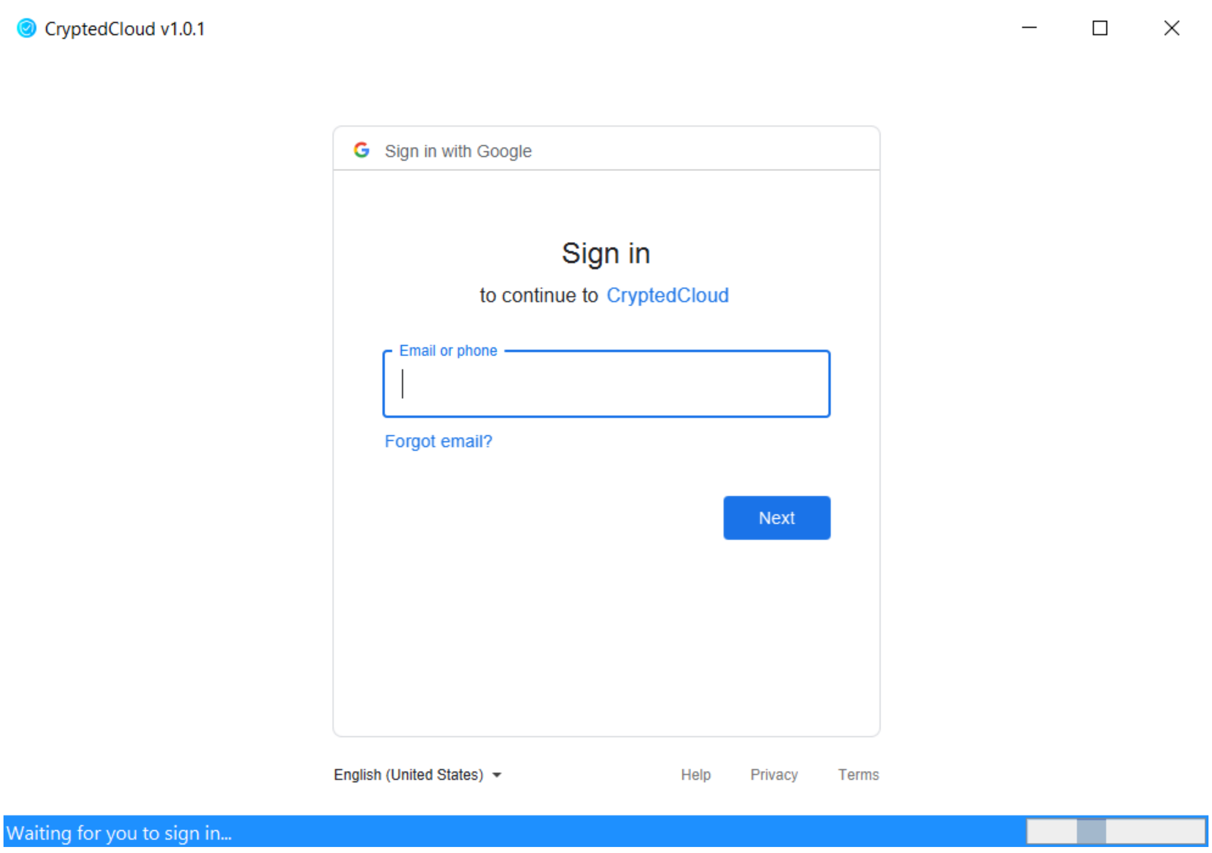
Figure 4.2: Configuration file containing JSON data

The second one is to keep abstraction as much as possible for the objects which allows us to have separation of concern among classes. This means, one class is less dependable to another class.

4.1.3 Functional Specification

In software development, functional specification specifies the functions that a system or component must perform. This documentation usually describes what is required to be done by the user and at the same time it also specifies what should be the inputs and outputs of an operation. In this section we will discuss how our CryptedCloud Client application works. This will cover all the parts of the user interface (UI) as well.

At first launch of CryptedCloud Client application, the user will be taken to the Google's sign in page. User needs to enter his/her email address which is associated with Google Drive account in order to continue. Google's sign in page is shown in Figure 4.3. Depending on the region of the user, the language of this page might be different. But the information required will remain the same.


CryptedCloud v1.0.1

Sign in with Google

Sign in

to continue to CryptedCloud

Email or phone

Forgot email?

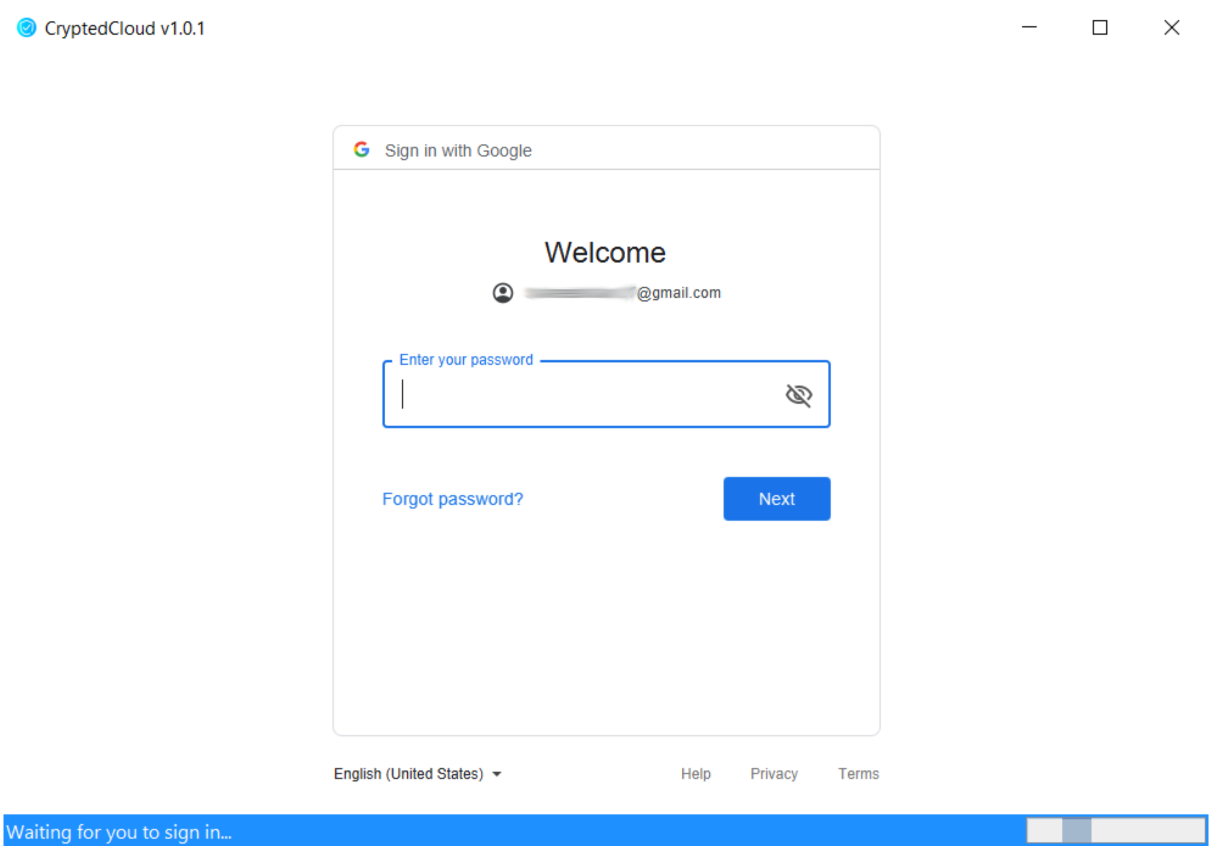
Next

English (United States)

Help Privacy Terms

Waiting for you to sign in...

Figure 4.3: Google's sign in page


CryptedCloud v1.0.1

Sign in with Google

Welcome

[redacted]@gmail.com

Enter your password

Forgot password?

Next

English (United States)

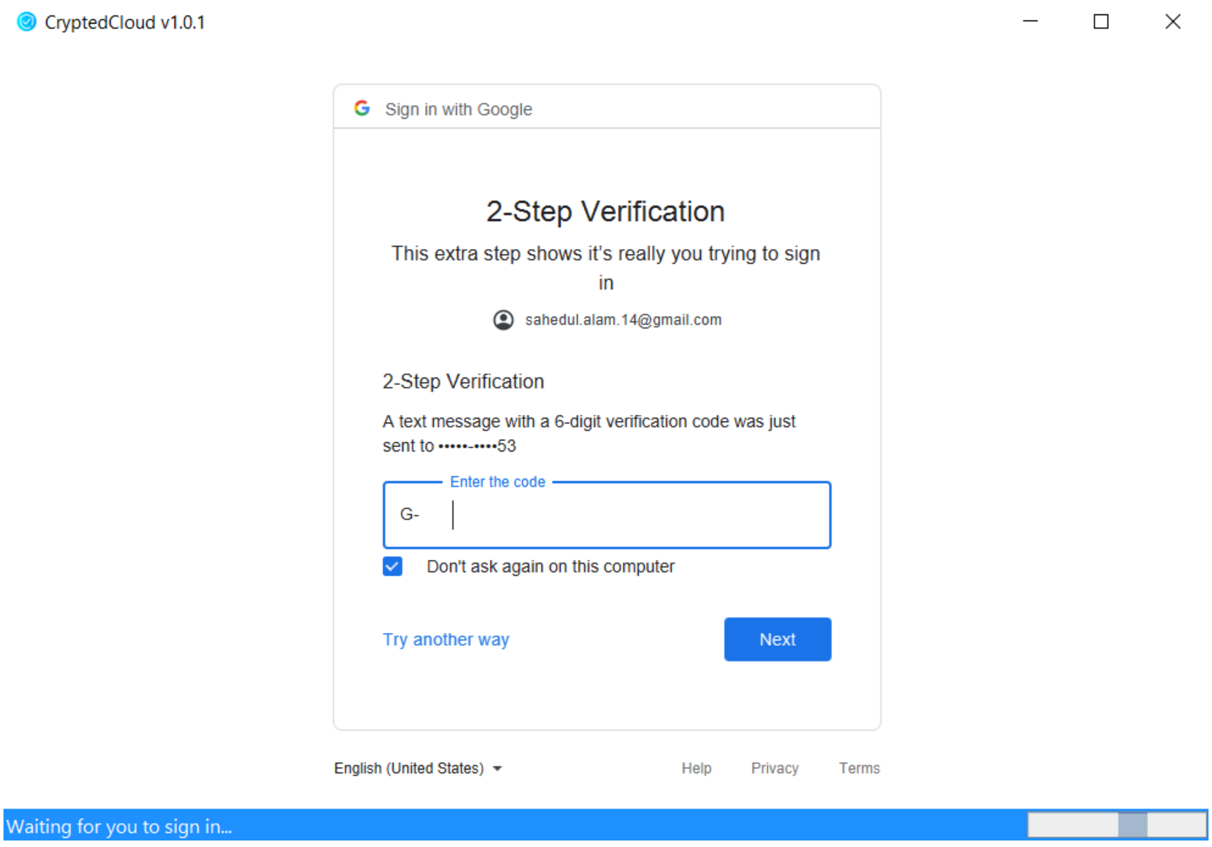
Help Privacy Terms

Waiting for you to sign in...

Figure 4.4: Password prompt for signing in

After entering the email address, user needs to click on “Next”. This will bring up the next page which will ask the user to enter the password. There is no chance of password hack as the password validation will take place in Google’s end and it will not be and cannot be traced by our application. The password prompt for sign in is shown in Figure 4.4.

If the user has two step authentication enabled in Google account, he/she will be prompted to enter the secret code sent to him/her via email or SMS (shown in Figure 4.5). This will also be performed by Google and is not related to CryptedCloud Client by any means.


CryptedCloud v1.0.1

Sign in with Google

2-Step Verification

This extra step shows it’s really you trying to sign in

sahedul.alam.14@gmail.com

2-Step Verification

A text message with a 6-digit verification code was just sent to •••••-••••53

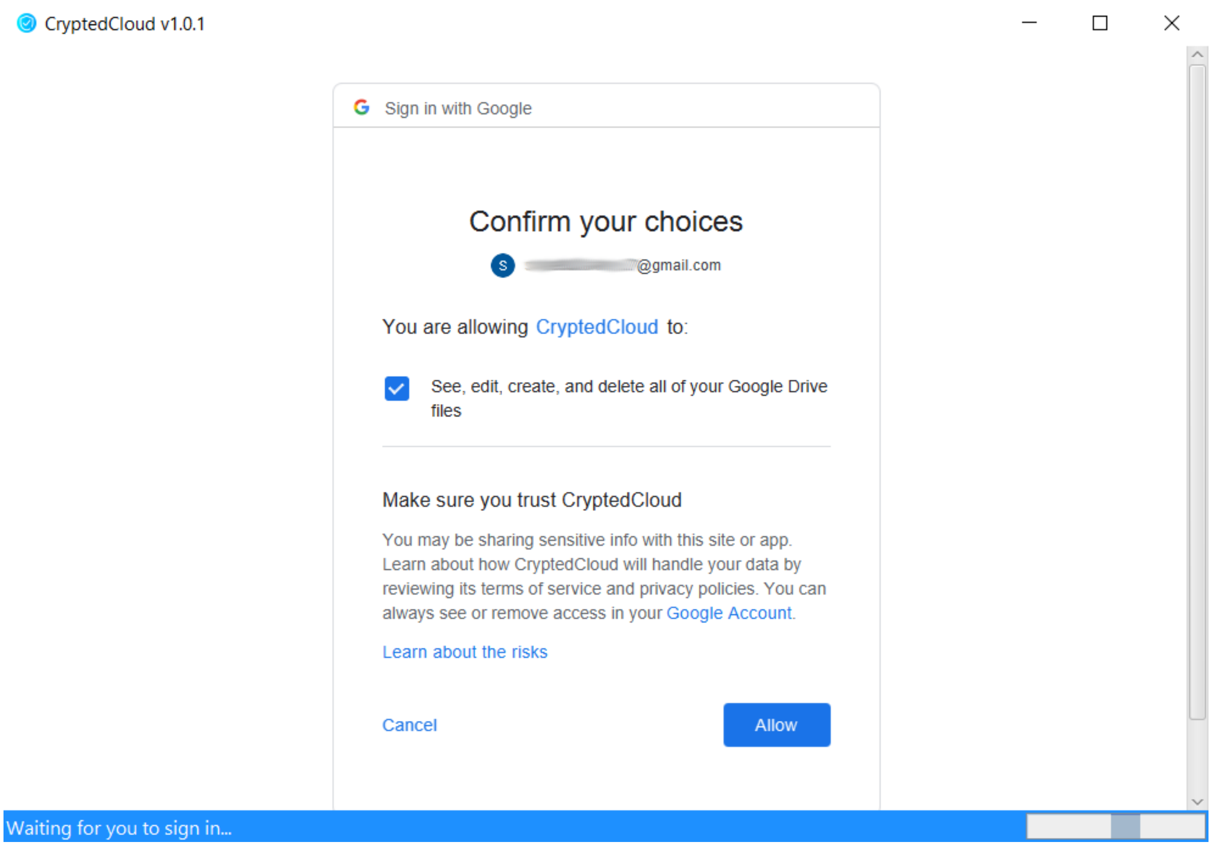
Enter the code

G-

☑ Don't ask again on this computer

Waiting for you to sign in...

Figure 4.5: Two step verification performed by Google


CryptedCloud v1.0.1

Sign in with Google

Confirm your choices

[redacted]@gmail.com

You are allowing CryptedCloud to:

✓ See, edit, create, and delete all of your Google Drive files

Make sure you trust CryptedCloud

You may be sharing sensitive info with this site or app. Learn about how CryptedCloud will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

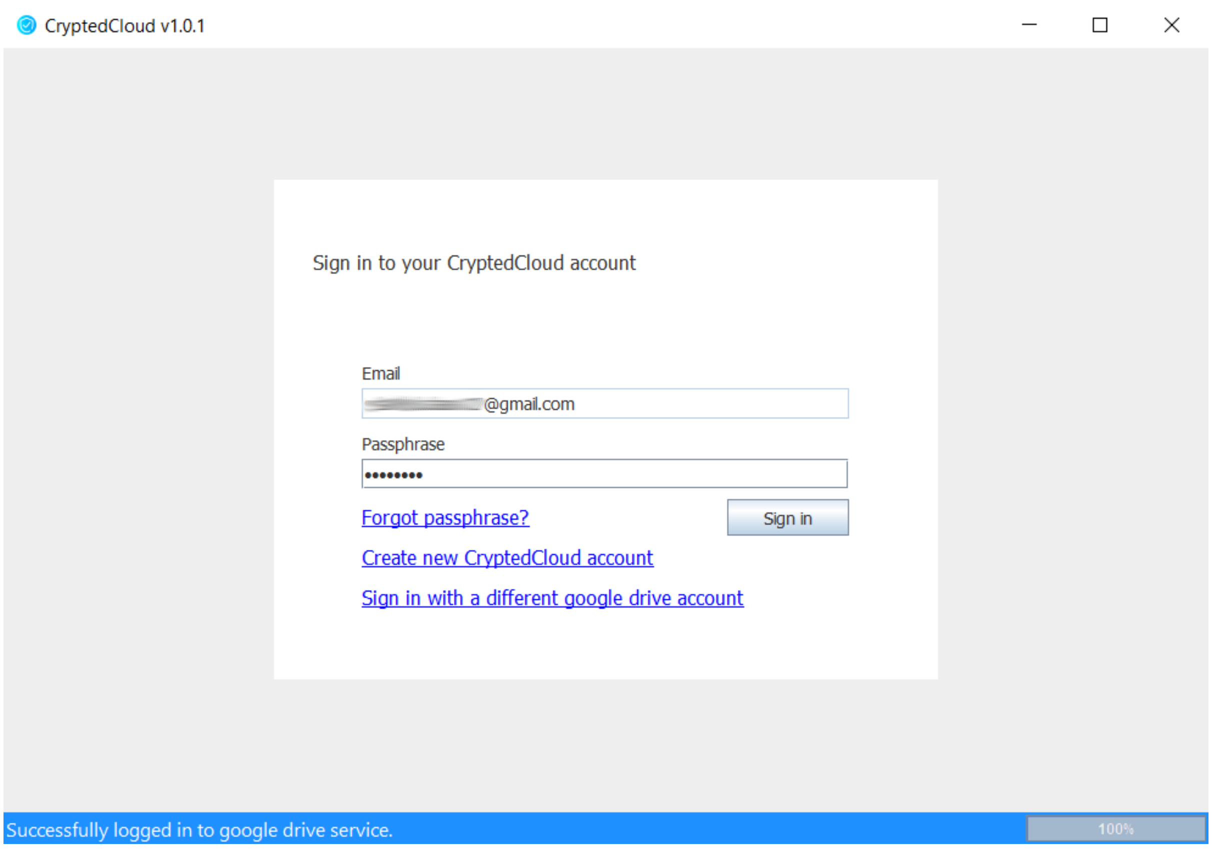
Waiting for you to sign in...

Figure 4.6: Grant CryptedCloud access to Google Drive

After the password is entered, the user needs to click on “Next”. It will ask the user to grant/allow CryptedCloud’s access to his/her Google Drive cloud storage (shown in Figure 4.6). The access to user’s cloud storage is required by the client application for performing upload, download, share and delete operation. None of these operations will be performed by the client application without user’s consent. In order to grant the access, user needs to make sure that the “See, edit, create and delete all of your Google Drive files” is checked and click “Allow”.

User will have to go through this Google Drive authentication only once. The Google Drive credentials will be stored safely (by Google Drive API v3) for later use. It is worth mentioning that, only the client application can access user’s cloud storage not the server side application.

Now the CryptedCloud Client will redirect to its own sign in page, shown in Figure 4.7. If user has already signed in to Google Drive previously, launching the client application will directly navigate to its own sign in page. In this page, user will see his/her email address already typed in the email field and it is not editable. There are two specific reasons behind this design choice. The most important reason is to make sure that the user does not have to type in the email address again. And secondly, to make sure that the email address used to access CryptedCloud is associated with cloud service which in our case Google Drive.


CryptedCloud v1.0.1

Sign in to your CryptedCloud account

Email
[redacted]@gmail.com

Passphrase
••••••••

Forgot passphrase?

Sign in

Create new CryptedCloud account

Sign in with a different google drive account

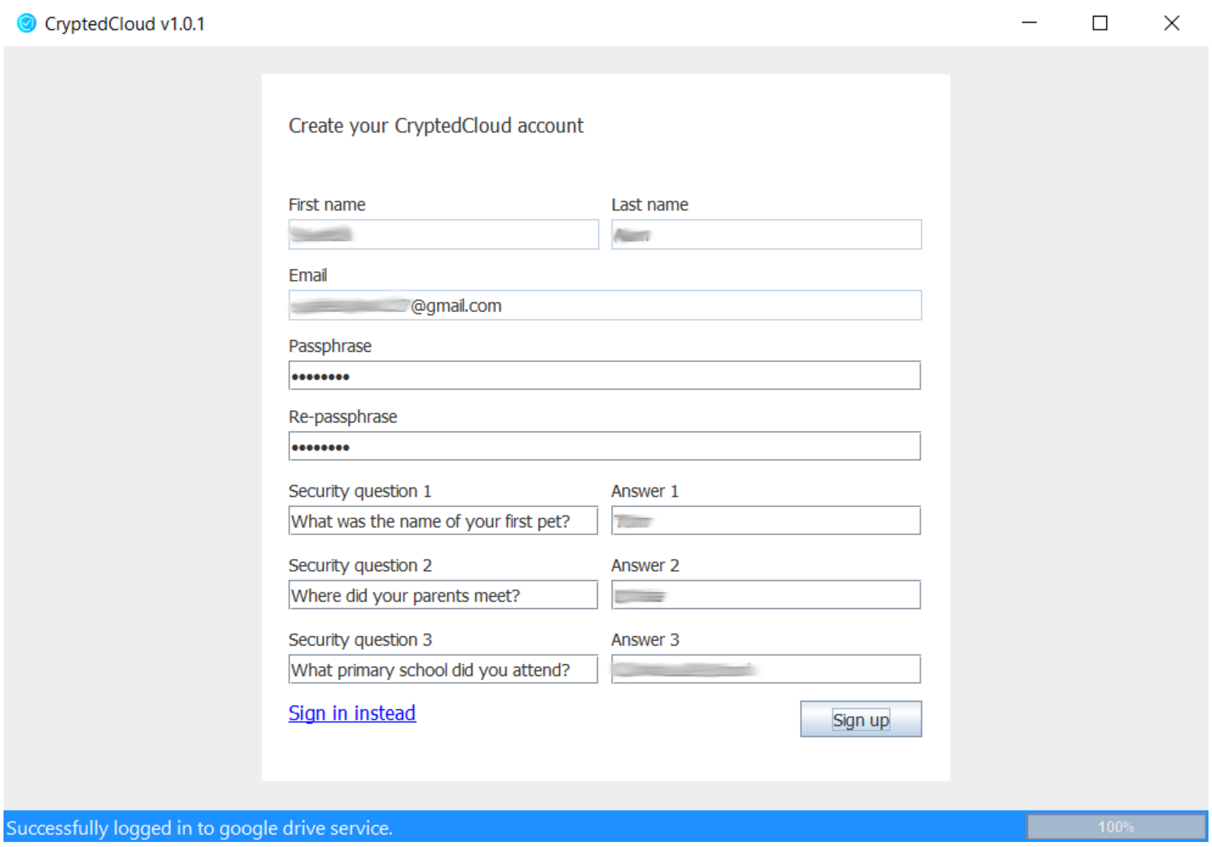
Successfully logged in to google drive service.

100%

Figure 4.7: CryptedCloud Client sign in page

At this stage, we are assuming that a user does not have a CryptedCloud account yet. So we will first discuss the registration page and then we will get back to the sign in page. A new account can be created by clicking the “Create new CryptedCloud account”. Landing to the account creation page, user will find the following information already placed in the specified fields. e.g. first name, last name, email, security question 1, 2 and 3. Information from the first name and last name field will not be sent to the server during the registration process. The email address field is read-only and already filled up with user’s email address. User will need to provide passphrase for his/her account. Passphrase must be at-least 8 characters long and both Passphrase and Re-passphrase must be the same. Otherwise, registration process will fail.

There are three more fields, Security questions 1, 2 and 3 which are for account recovery purpose as mentioned in Section 3.1 of Chapter 3. These fields are optional for user to fill up but we highly encourage to provide this information. The questions can be modified but we have provided some sample questions which can be used as well. If the answer is not provided for any question, that question will be ignored and will not be added to the user’s recovery information. Answers are not case sensitive as mentioned previously. Starting and trailing whitespaces are ignored as well. After the mandatory information are provided, user needs to click on “Sign up” button.


CryptedCloud v1.0.1

Create your CryptedCloud account

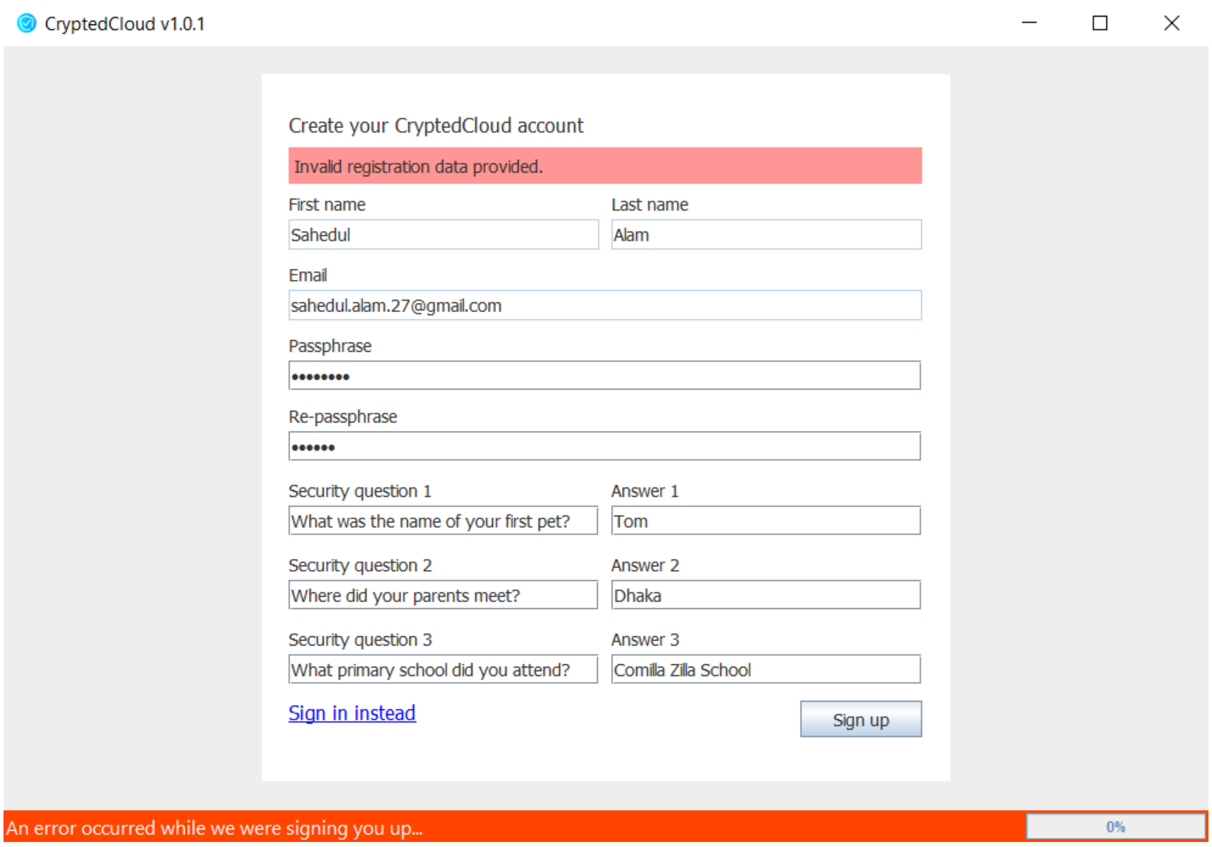
First name	Last name
[redacted]	[redacted]
Email	
[redacted]@gmail.com	
Passphrase	
••••••••	
Re-passphrase	
••••••••	
Security question 1	Answer 1
What was the name of your first pet?	[redacted]
Security question 2	Answer 2
Where did your parents meet?	[redacted]
Security question 3	Answer 3
What primary school did you attend?	[redacted]

Sign in instead

Sign up

Successfully logged in to google drive service. 100%

Figure 4.8: CryptedCloud Client account creation page


CryptedCloud v1.0.1

Create your CryptedCloud account

Invalid registration data provided.

First name	Last name
Sahedul	Alam
Email	
sahedul.alam.27@gmail.com	
Passphrase	
••••••••	
Re-passphrase	
••••••	
Security question 1	Answer 1
What was the name of your first pet?	Tom
Security question 2	Answer 2
Where did your parents meet?	Dhaka
Security question 3	Answer 3
What primary school did you attend?	Comilla Zilla School

Sign in instead

Sign up

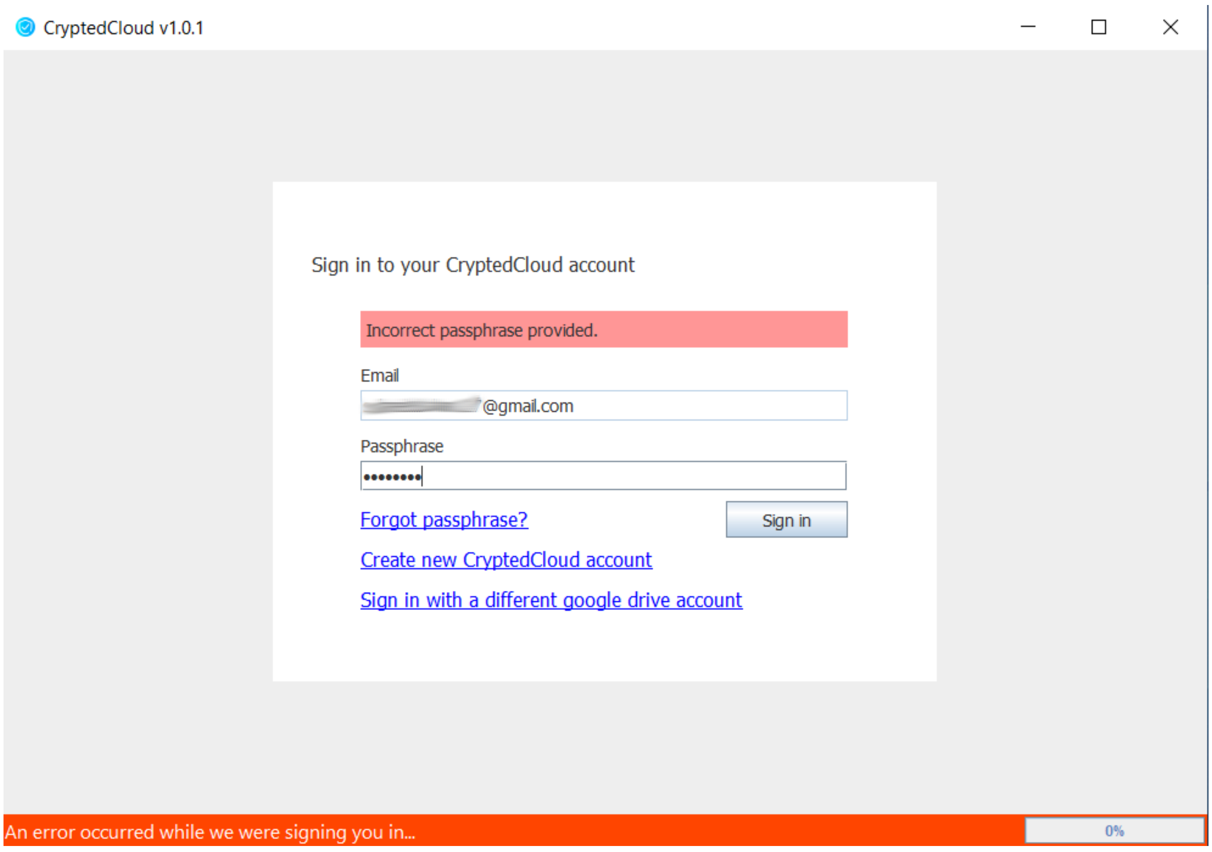
An error occurred while we were signing you up... 0%

Figure 4.9: Error occurred during CryptedCloud account creation

If erroneous input detected in the fields during account creation, an error message will be shown in the notification area as well as in the status bar. e.g. if the passphrase provided is less than 8 characters or Passphrase and Re-passphrase do not match, the client application will not validate the input and will generate error. If a user is already registered and attempts to re-register, it would also generate error. Figure 4.9 shows the error condition during account creation. Upon successful registration, user will be redirected to the CryptedCloud sign in page.

As of now, we assume that the user has a CryptedCloud account. So, let us get back to the discussion regarding sign in page again. User needs to provide his/her CryptedCloud passphrase and click "Sign in" to access all the features of CryptedCloud. This is the passphrase the user has set during his/her account creation on CryptedCloud, not the Google Drive password.

If user enters incorrect passphrase, an error message will be shown in the notification area. More details regarding the error can be found in the status bar at the bottom. The status bar will turn "Orange" indicating that an error has occurred. The erroneous condition is shown above in Figure 4.10.


CryptedCloud v1.0.1

Sign in to your CryptedCloud account

Incorrect passphrase provided.

Email

[redacted]@gmail.com

Passphrase

••••••••

Forgot passphrase?
Sign in
Create new CryptedCloud account
Sign in with a different google drive account

An error occurred while we were signing you in...

Figure 4.10: Sign in error due to incorrect passphrase

Upon successful sign in, user will be taken to the file explorer and will be able to access all the functionalities of CryptedCloud. From the file explorer, user can choose whichever file he/she wants to download, share or delete. The file explorer is shown in Figure 4.1.

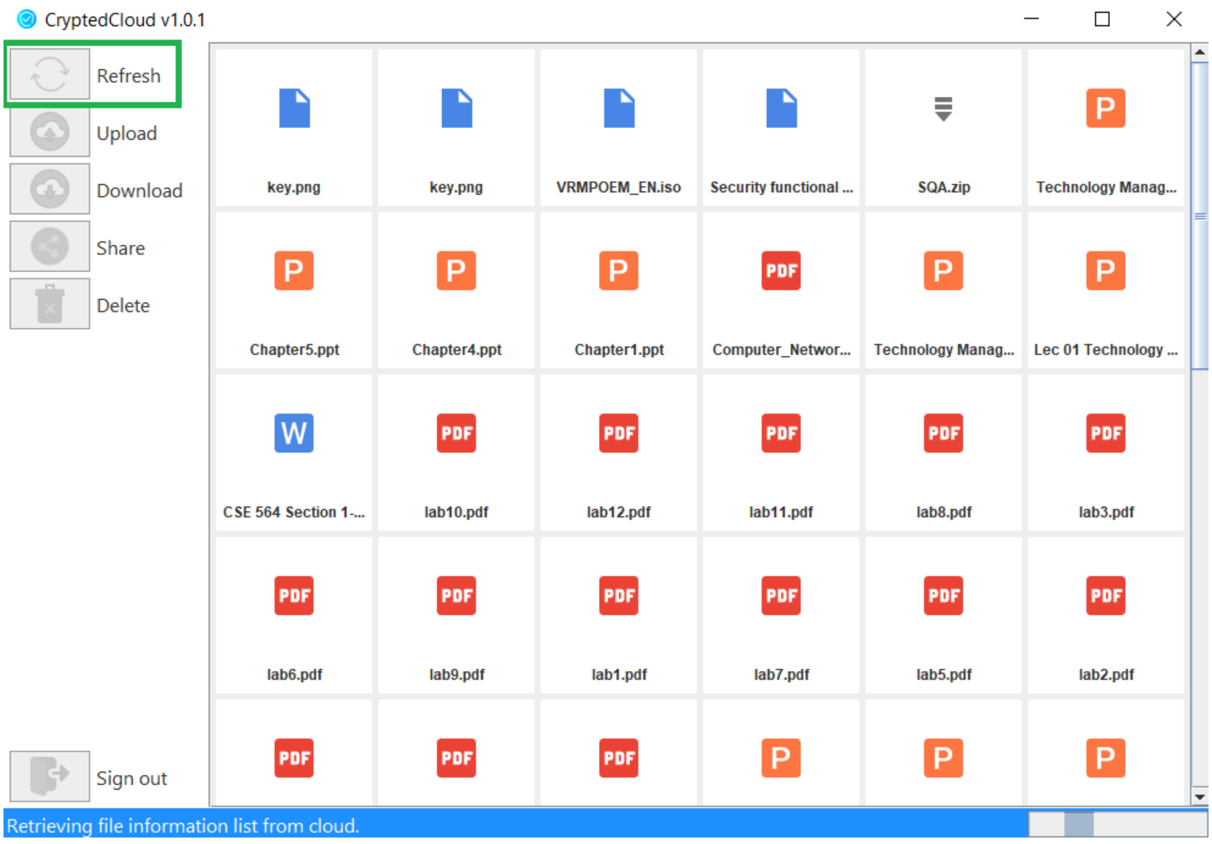
As the user is logged in successfully by now, user will find six different buttons to perform six individual operations. Those are-

1. Refresh
2. Upload
3. Download
4. Share
5. Delete
6. Sign out

The functionalities of all these buttons are self explanatory. Still we will describe the use cases of all these buttons and how these buttons work one by one.

Refresh

Clicking on “Refresh” button will retrieve the latest changes added/deleted files from Google Drive. As there are numerous ways of adding/deleting files to Google Drive e.g. via web browsers, mobile apps, etc. Figure 4.11 shows the refresh operation being performed by CryptedCloud Client.



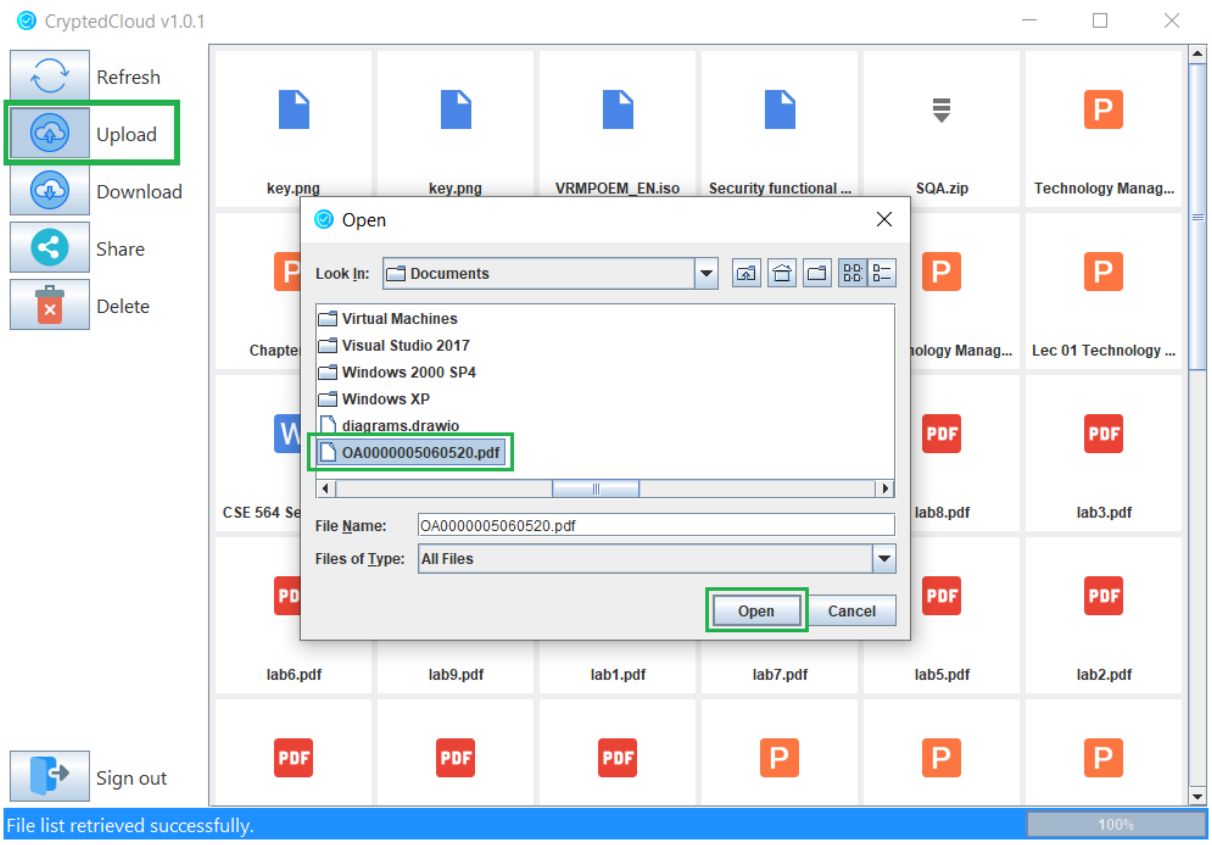
Retrieving file information list from cloud.

Figure 4.11: Refresh operation is performed

While the refresh operation is being performed, all the buttons will be disabled and the user will be able to see the progress status on the status bar at the bottom. After a successful refresh operation, status bar will show the message, “File list retrieved successfully”.

Upload

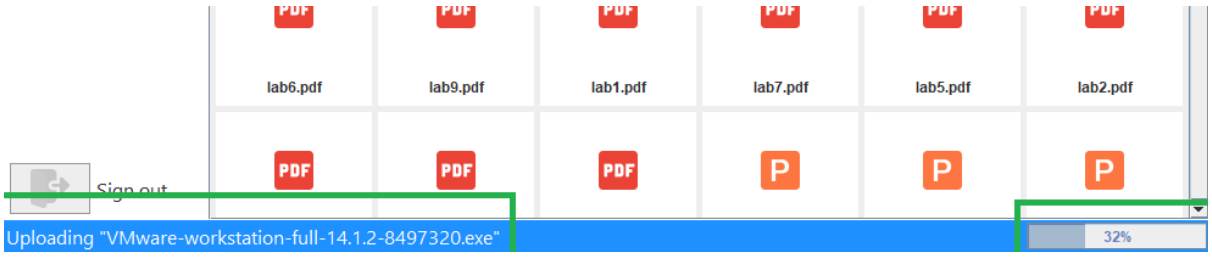
Uploading files with our CryptedCloud Client is just as easy as uploading files using any other application. User just needs to click the “Upload” button. A file selection dialog box will popup. Here user needs to select the file that needs to be uploaded.



File list retrieved successfully.

Figure 4.12: File upload using CryptedCloud client

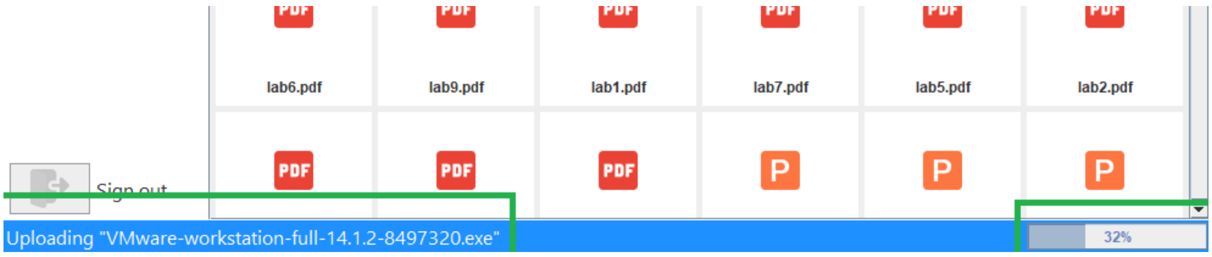
Then clicking on “Open” button will initiate the upload process. As Google Drive API v3 does not provide access to file upload stream, we were unable to combine encrypt and upload operation. Thus, the CryptedCloud Client will encrypt the selected file in a temporary location (shown in Figure 4.13).



Uploading "VMware-workstation-full-14.1.2-8497320.exe"

Figure 4.13: Encryption in progress

After the file encryption is completed, the client application will start uploading the file to the cloud. All these will be done without any intervention of the user. User can see the progress in the status bar (shown in Figure 4.14). During upload operation, all the other buttons will remain disabled.



Uploading "VMware-workstation-full-14.1.2-8497320.exe"

Figure 4.14: Upload in progress

After the upload operation is completed successfully, refresh operation will be performed to validate the latest changes in Google Drive strogate. Thus, the file will be available in file explorer (shown in Figure 4.15).

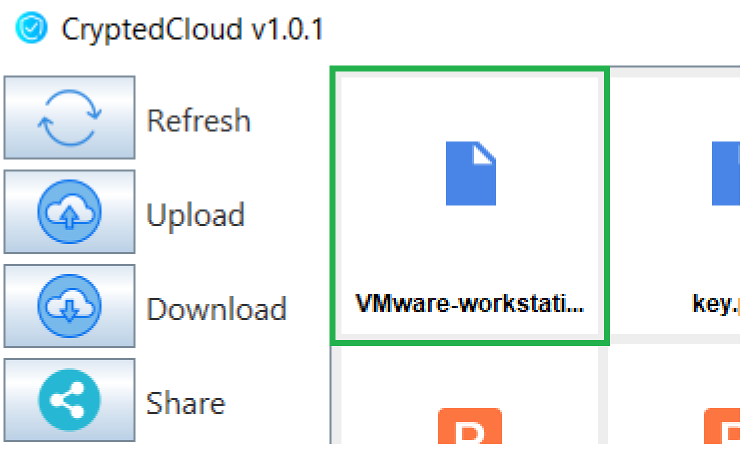
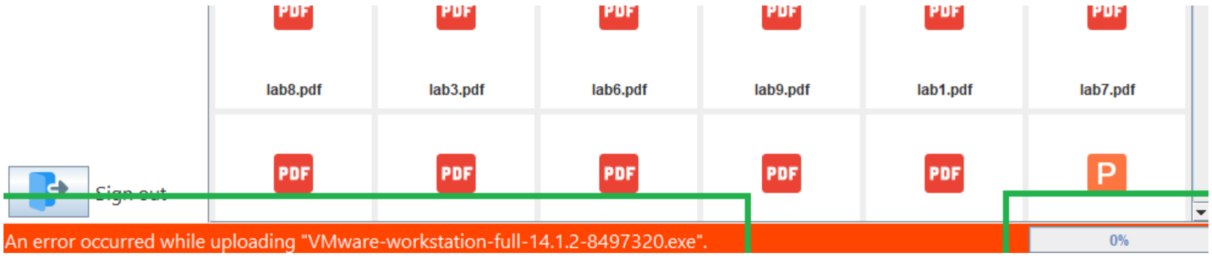


Figure 4.15: Uploaded file is shown in file explorer

If for some reason upload operation fails e.g. user access to the specified file was revoked by the owner or there might be internet connectivity issues etc., an error message is shown in the status bar and the status bar will turn Orange (shown in Figure 4.16).

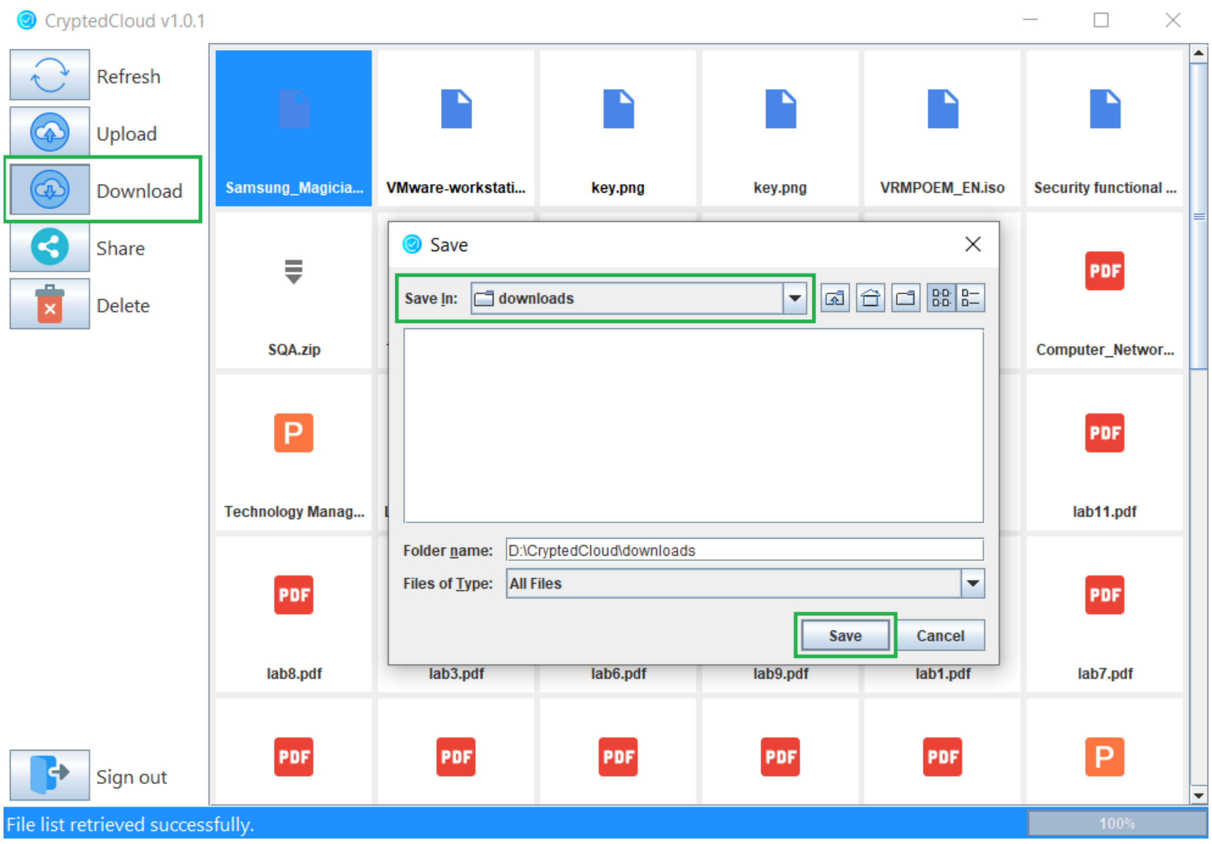


An error occurred while uploading "VMware-workstation-full-14.1.2-8497320.exe".

Figure 4.16: Upload operation failed

Download

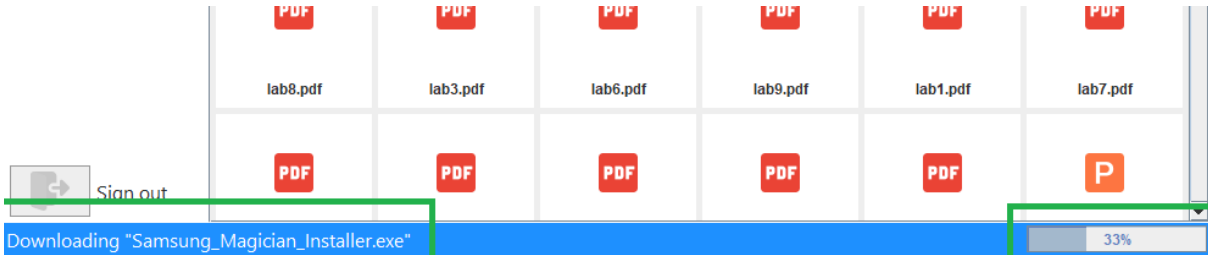
Files can be downloaded very easily using CryptedCloud Client. All the user needs to do is select the file from cloud storage and click on "Download" button. It will bring up a dialog box. In that dialog box, user needs to specify the location where the file will be downloaded (shown in Figure 4.17).



File list retrieved successfully.

Figure 4.17: File download using CryptedCloud client

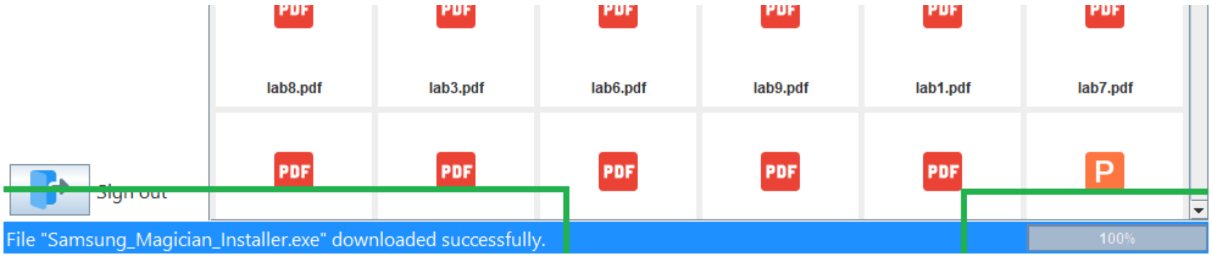
After selecting the location, user needs to click on "Save" button and the download operation will be initiated (shown in Figure 4.18). User can see the progress of the download operation in the status bar.



Downloading "Samsung_Magician_Installer.exe"

Figure 4.18: Download in progress

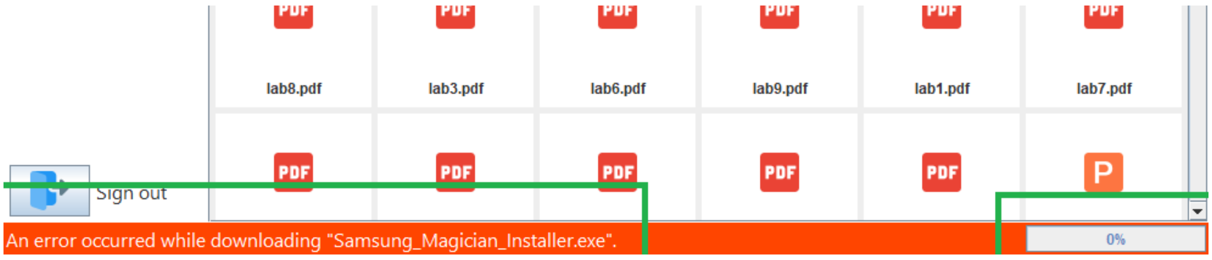
Unlike upload operation, Google Drive API v3 provides direct access to the download stream. So downloading and decrypting can be performed simultaneously. So, once the download operation is successful, the file is also decrypted and ready to use. Upon successful download, status bar will notify the user with a success message (shown in Figure Figure 4.19).



File "Samsung_Magician_Installer.exe" downloaded successfully.

Figure 4.19: Download succeeded

Just like upload operation, there might be certain cases when download operation will fail. Most obvious one would be due to internet connectivity issues. In these cases, the user will be shown an error message in the status bar (shown in Figure 4.20).



An error occurred while downloading "Samsung_Magician_Installer.exe".

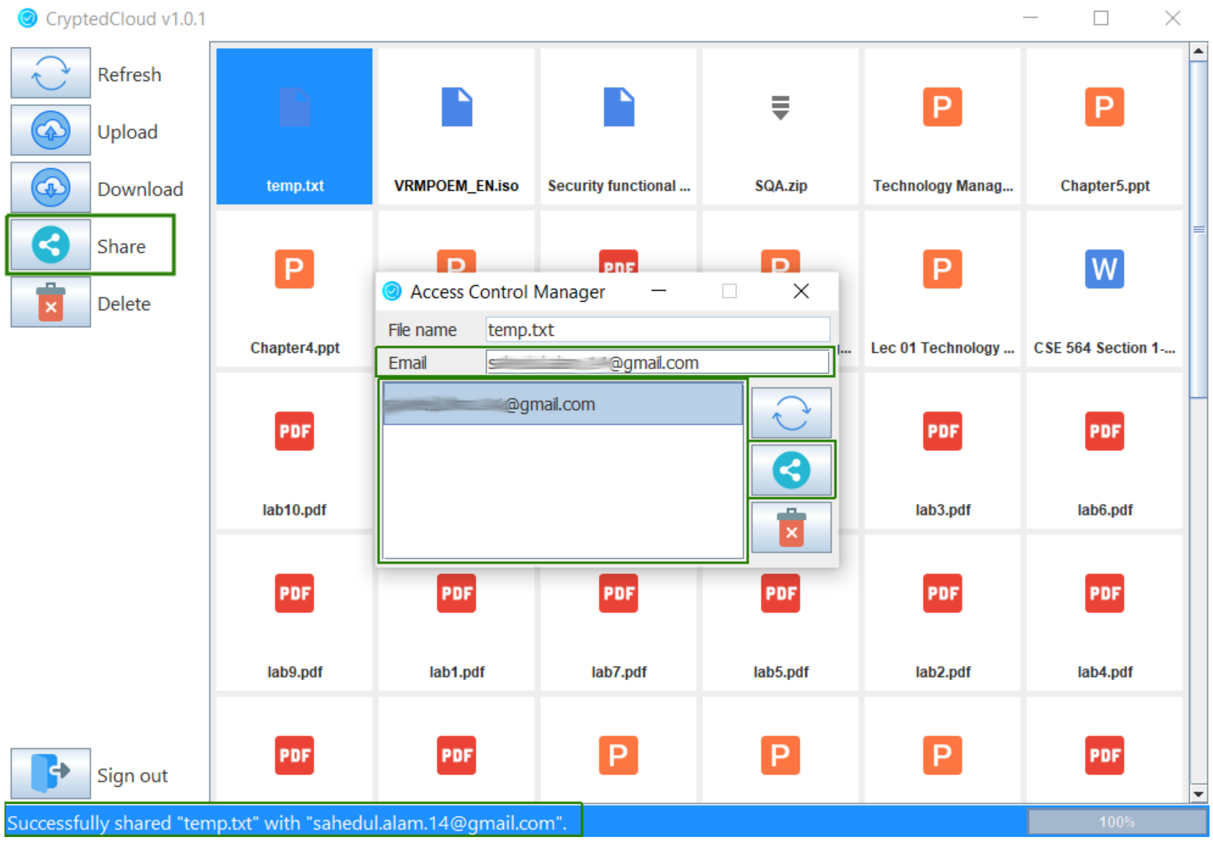
Figure 4.20: Download operation failed

Share

CryptedCloud Client also allows sharing a file to another user. To share a file, the user needs to select a file from the file explorer and click on "Share" button. This will bring up the Access Control Manager dialog box. There are two text fields, a list and three buttons. The text field at the top of the dialog box will display the selected file name. The second text field takes email address of the person with whom the file needs to be shared. So the user needs to enter the email address of the other person and click on the "Share" button (the second button).

During the file sharing process, user will be asked to wait for the operation to complete. In the meantime, all the complex cryptographic operations will be performed in the background by the CryptedCloud Client. If the entered email address is associated with CryptedCloud account, the file will be shared successfully. Status bar will display a success message. A successful file sharing is shown in Figure 4.21.

But if the provided email address user does not belong to CryptedCloud account, the client application will display an error message (shown in Figure 4.22). The CryptedCloud Server will send an email to the specified email address stating that the user wants to share a secret file with him and to receive the file, he/she must register on CryptedCloud service.



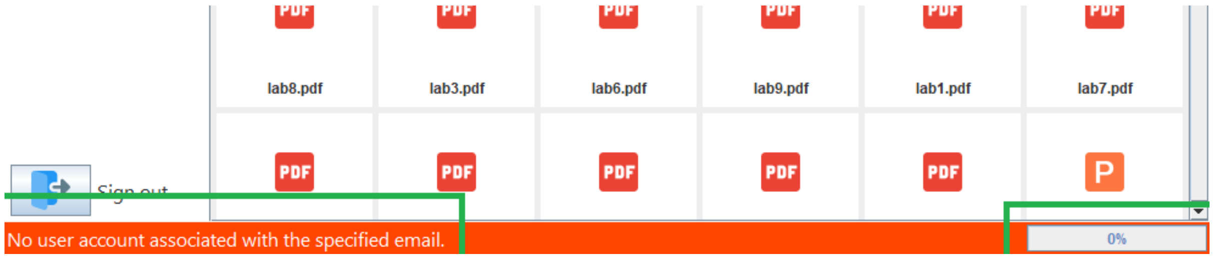
File name: temp.txt

Email: [redacted]@gmail.com

[redacted]@gmail.com

Successfully shared "temp.txt" with "sahedul.alam.14@gmail.com".

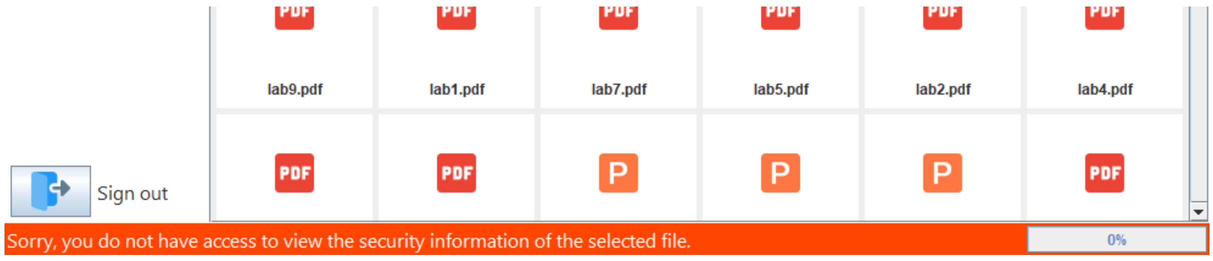
Figure 4.21: Sharing file using CryptedCloud Client



No user account associated with the specified email.

Figure 4.22: File sharing failed

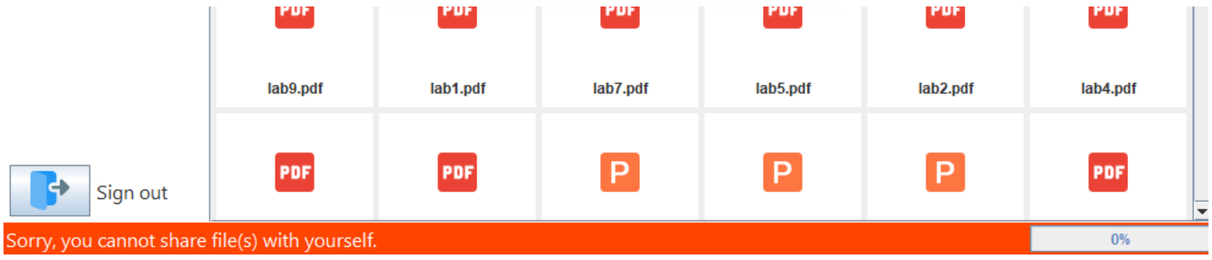
If the user is not the owner of the file, clicking on the “Share” button on the left hand side of the main window of the CryptedCloud Client, the Access Control Manager dialog box will not open. Thus, an error message is generated as shown in Figure 4.23.



Sorry, you do not have access to view the security information of the selected file.

Figure 4.23: Error occurred while opening Access Control Manager

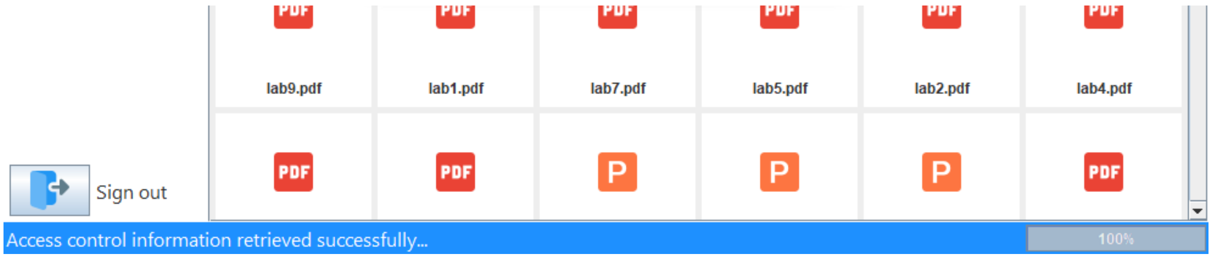
In the email field, if user types in his/her own email address, an error message (shown in Figure 4.24) will be generated.



Sorry, you cannot share file(s) with yourself.

Figure 4.24: Sharing fails when own email address is provided

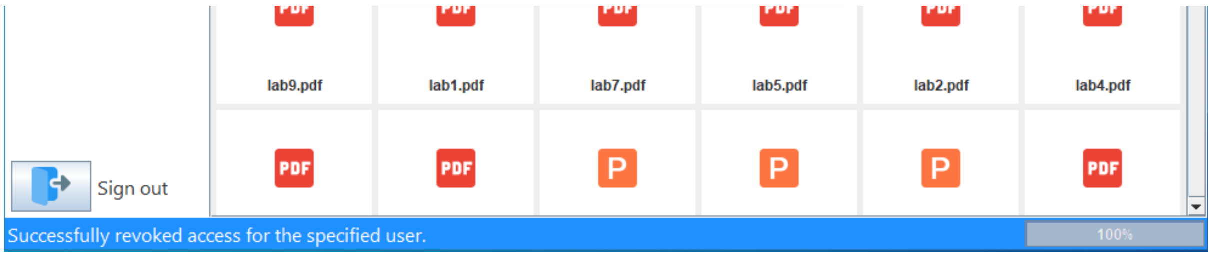
The top most button ("Refresh" button) in the Access Control Manager dialog box can be used to refresh the Access Control List (shown in Figure 4.21). It fetches the email address of the users with whom a file has been shared and displays on the list. The refresh operation is automatically performed whenever Access Control Dialog box is opened. The button can be used for performing refresh operation manually. Upon successful completion of refresh operation, a success message will be displayed in the status bar (shown in Figure 4.25).



Access control information retrieved successfully...

Figure 4.25: Message stating successful refresh operation

Now, it might happen that a user wants to revoke access of a particular user. In that case, the third button ("Delete" button) will be useful. To do so, the user needs to select the email address of the user (whose access needs to be revoked) from the Access Control List. After selecting an email, user needs to click on the "Delete" button. When the access revocation is successful, that particular email address will be removed from the list and the user will be notified through a success message (shown in Figure 4.26).

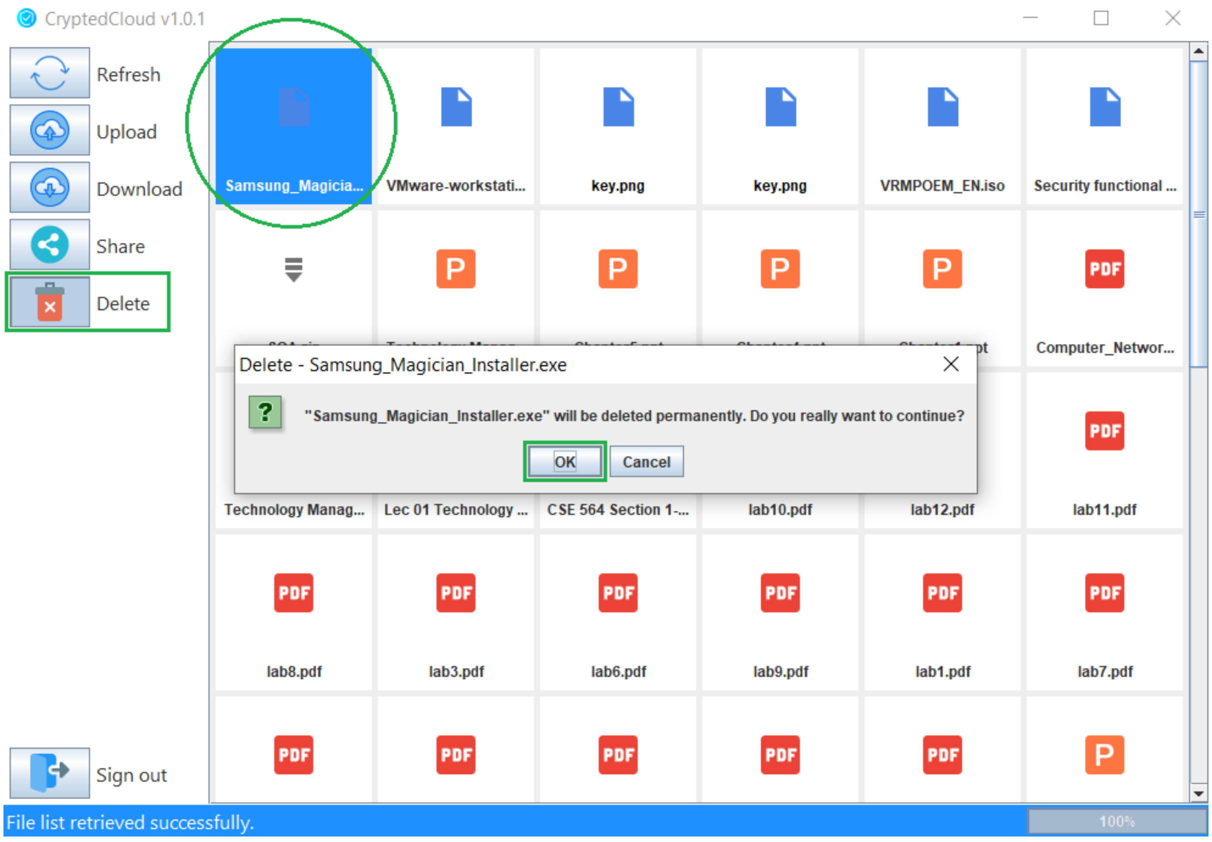


Successfully revoked access for the specified user.

Figure 4.26: Successfully revoked user access

Delete

To delete a file, user needs to select a file in the file explorer and click on “Delete” button. A prompt will popup to confirm if the user wants to delete the file (shown in Figure 4.24).



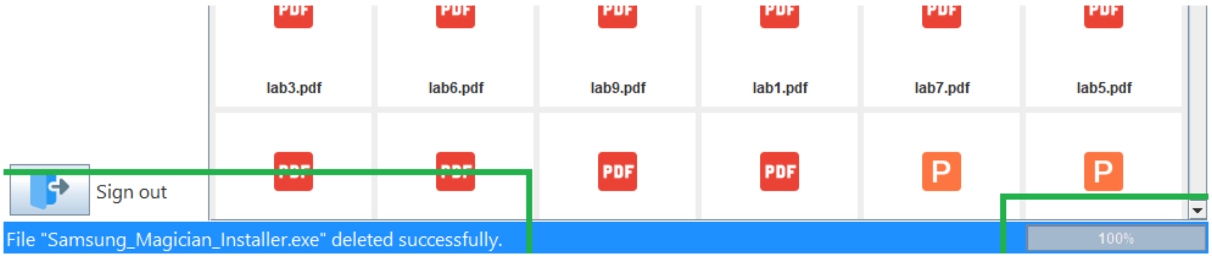
Delete - Samsung_Magician_Installer.exe

"Samsung_Magician_Installer.exe" will be deleted permanently. Do you really want to continue?

File list retrieved successfully.

Figure 4.24: File delete using CryptedCloud Client

If the user has proper access rights or in other words if the user is the owner of the file, clicking the “OK” button will delete the file from cloud storage. Upon successful deletion, user will receive success notification on the status bar (shown in Figure 4.25).



File "Samsung_Magician_Installer.exe" deleted successfully.

Figure 4.25: File deleted successfully

The selected file will also be removed from the file explorer (shown in Figure 4.26). If the file was shared with other users, they will no longer be able to access the file.

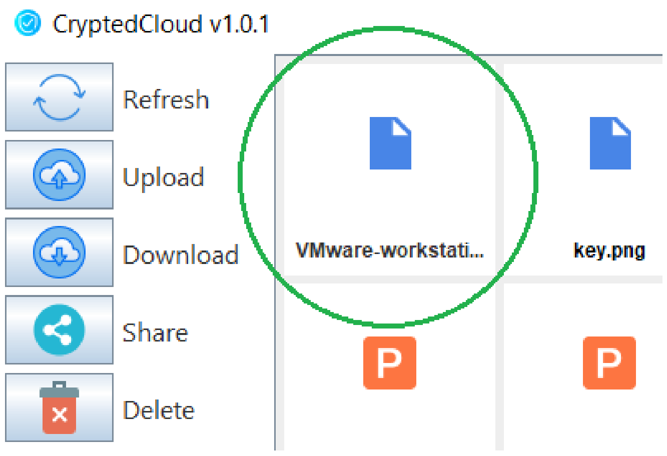
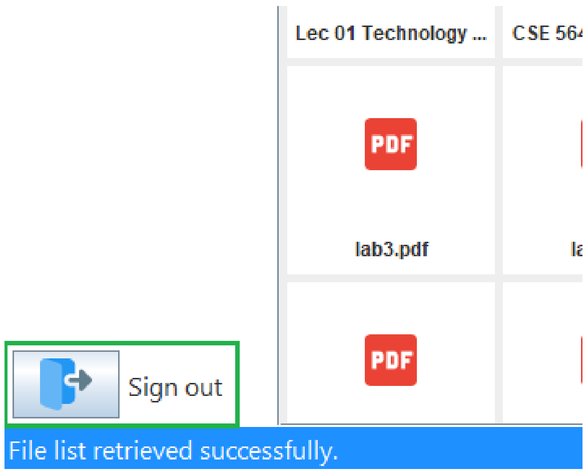


Figure 4.26: File removed from the file explorer

If the user requesting for deletion is not the owner of the selected file, the file will not be removed from the cloud storage. Instead, it will remove the user's access from that particular file and the user will no longer find the file in the file explorer.

Sign out

This is the last button in this column. If the user wants to sign out from the CryptedCloud account, just clicking on the "Sign out" button (shown in Figure 4.28) will take the user back to the CryptedCloud sign in page (shown in Figure 4.7).



File list retrieved successfully.

Figure 4.28: Sign out from CryptedCloud account

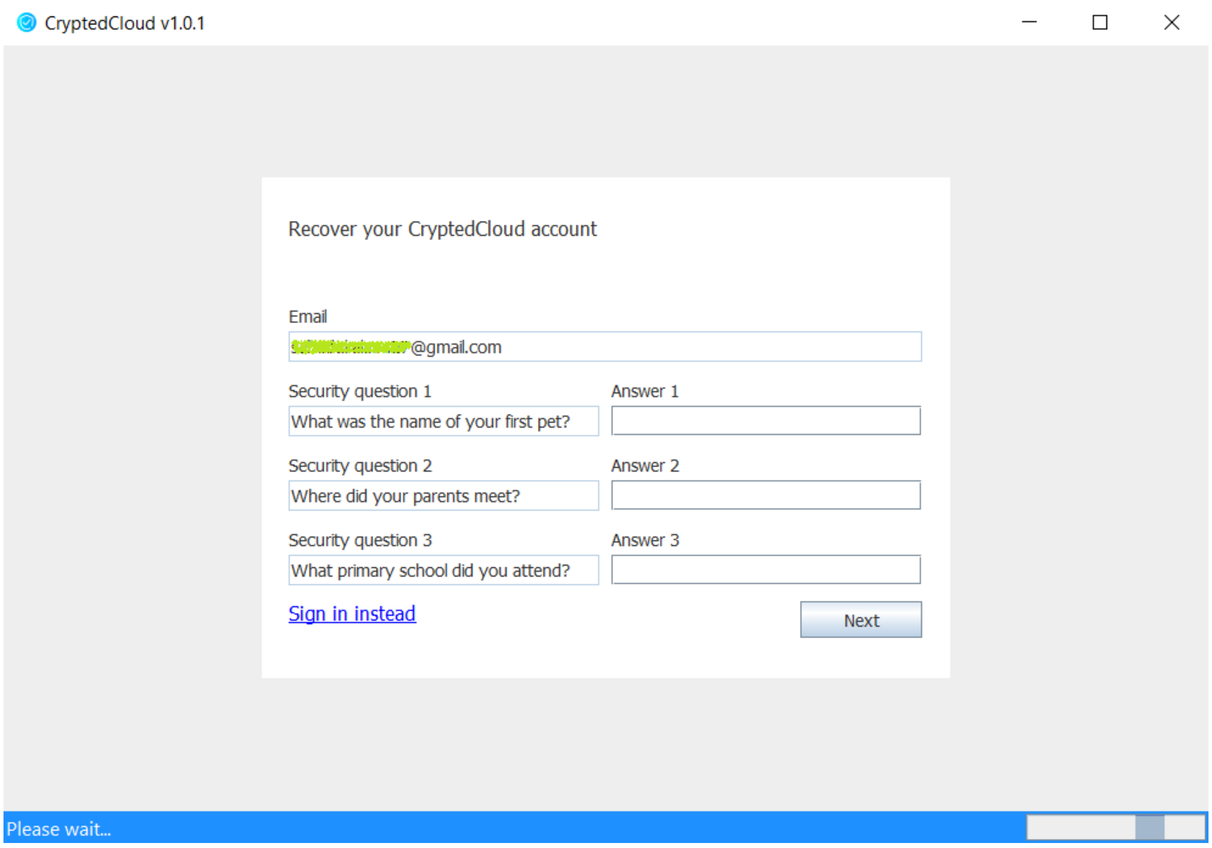
Now, within the sign in screen of CryptedCloud Client, there are two more options-

a) Sign in with a different google drive account, and
b) Forgot passphrase?

As we have mentioned previously, Google Drive credentials will be stored in client device so that the user does not have to provide the credentials every time the client application is opened. But what if the user has multiple Google Drive accounts and he/she wants to switch to another account? For that reason, we have this option to Sign in with a different Google

Drive account (shown in Figure 4.7 below "Create new CryptedCloud account"). Clicking on this will take the user back to the Google's sign in page (shown in Figure 4.3).

The second option, "Forgot passphrase?" (shown in Figure 4.7 above "Create new CryptedCloud account") is provided for account recovery. It is very likely for users to forget their passphrases. If a user has provided recovery information during account creation, clicking on this option will open the account recovery page (shown in Figure 4.29)


CryptedCloud v1.0.1

Recover your CryptedCloud account

Email
[redacted]@gmail.com

Security question 1
What was the name of your first pet?

Answer 1

Security question 2
Where did your parents meet?

Answer 2

Security question 3
What primary school did you attend?

Answer 3

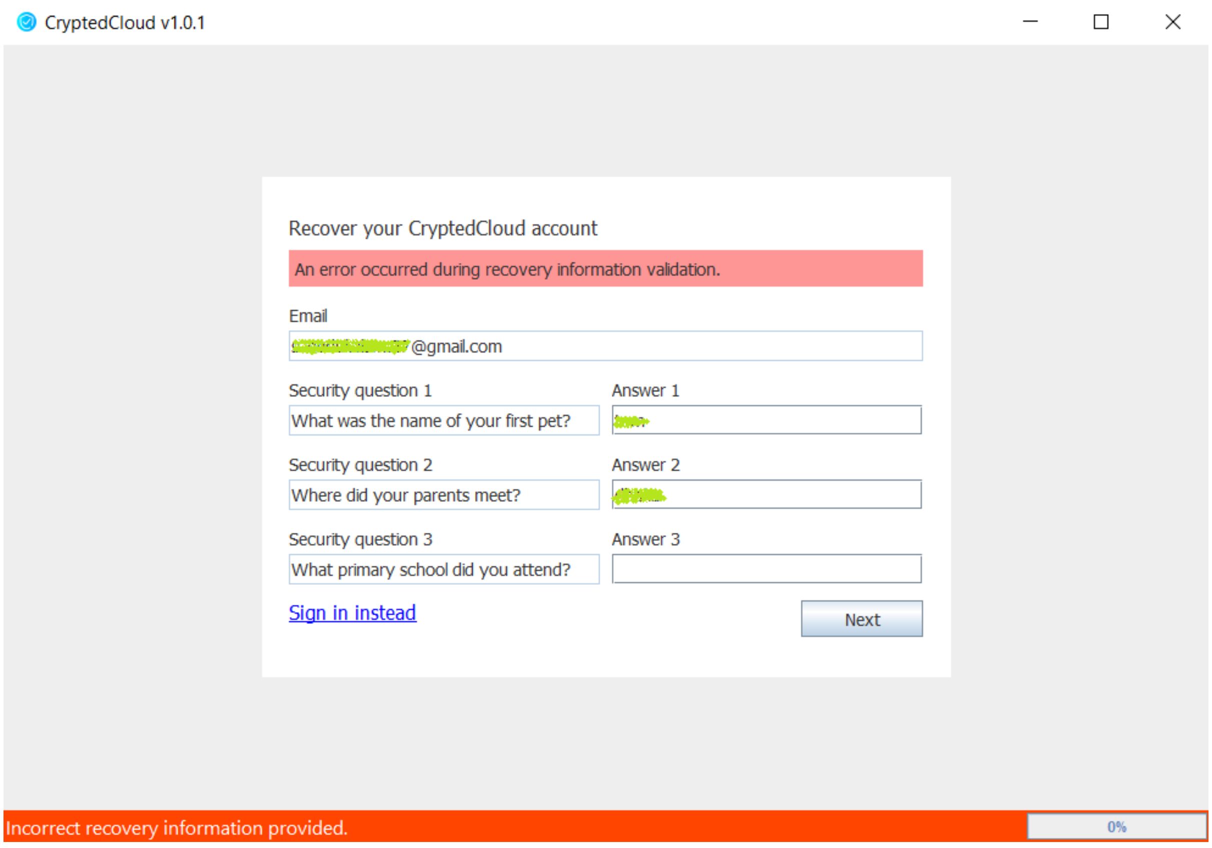
Sign in instead

Next

Please wait...

Figure 4.29: Account recovery page

User needs to provide the correct answers to the questions and click on "Next" button. The user will receive an error message if incorrect answers are provided (shown in Figure 4.30).


CryptedCloud v1.0.1

Recover your CryptedCloud account

An error occurred during recovery information validation.

Email: [redacted]@gmail.com

Security question 1: What was the name of your first pet? | Answer 1: [redacted]

Security question 2: Where did your parents meet? | Answer 2: [redacted]

Security question 3: What primary school did you attend? | Answer 3:

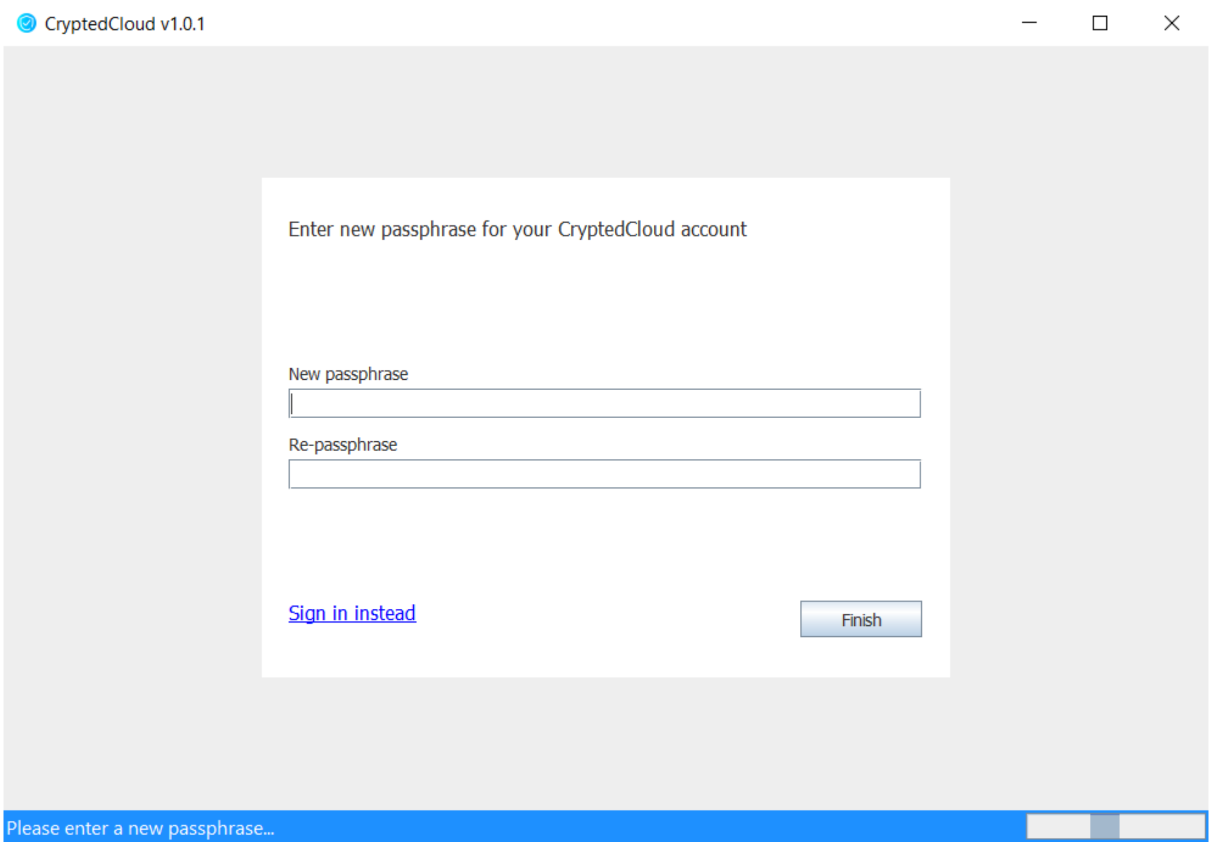
Incorrect recovery information provided.

Figure 4.30: Error occurred due to incorrect recovery information

If all the answers are correct, the user will be taken to the next section where user will be asked to provide a new passphrase (shown in Figure 4.31). The new passphrase must also match the CryptedCloud passphrase criteria which is, passphrase must be at least 8 characters long. Otherwise passphrase recovery will fail (shown in Figure 4.32).

If both Passphrase and Re-passphrase are provided correctly, the account recovery will succeed. User will be taken back to the CryptedCloud sign in page (shown in Figure 4.7).

Now, if the user does not have any account recovery information associated with his/her account or in other words, if the user had not provided any Security question or corresponding answer, clicking on “Forgot passphrase?” will show an error message stating the reason (shown in Figure 4.33).


CryptedCloud v1.0.1

Enter new passphrase for your CryptedCloud account

New passphrase

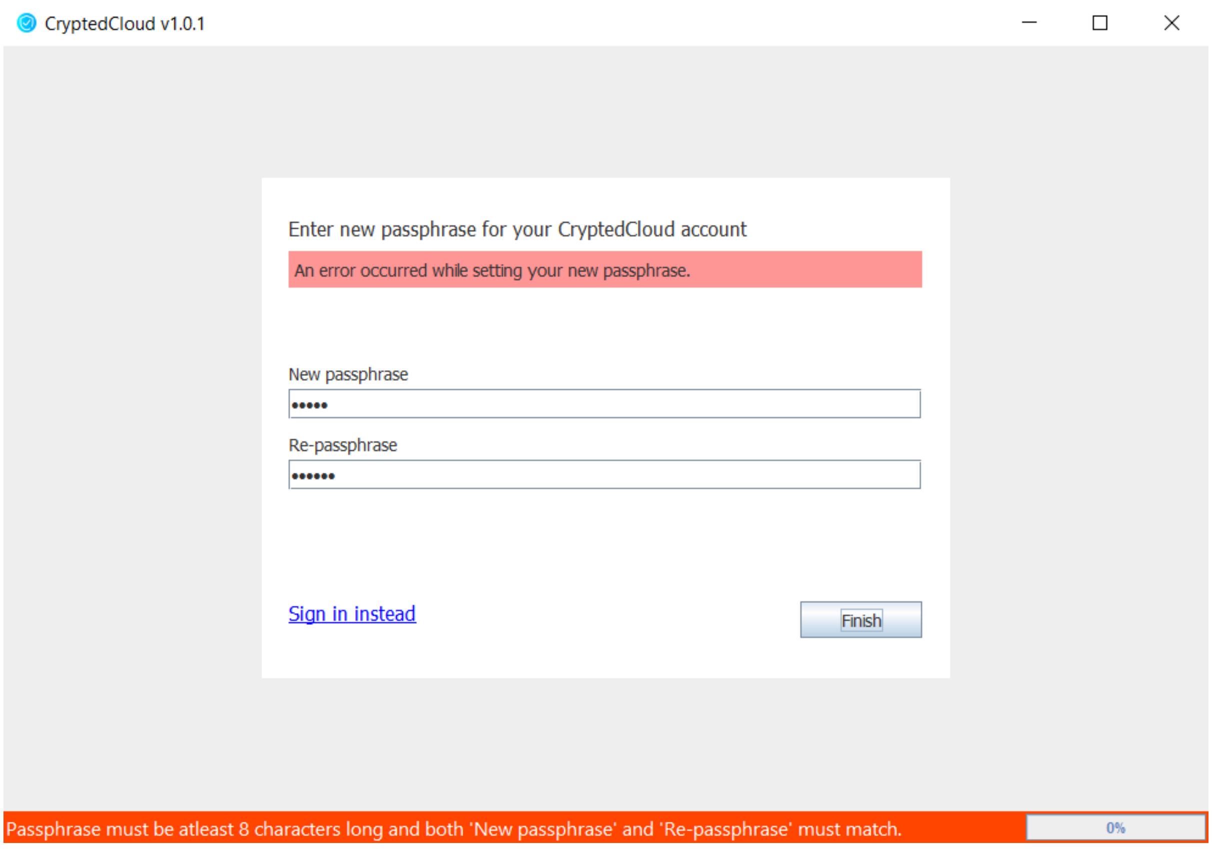
Re-passphrase

Sign in instead

Finish

Please enter a new passphrase...

Figure 4.31: Setting up new passphrase


CryptedCloud v1.0.1

Enter new passphrase for your CryptedCloud account

An error occurred while setting your new passphrase.

New passphrase

•••••

Re-passphrase

••••••

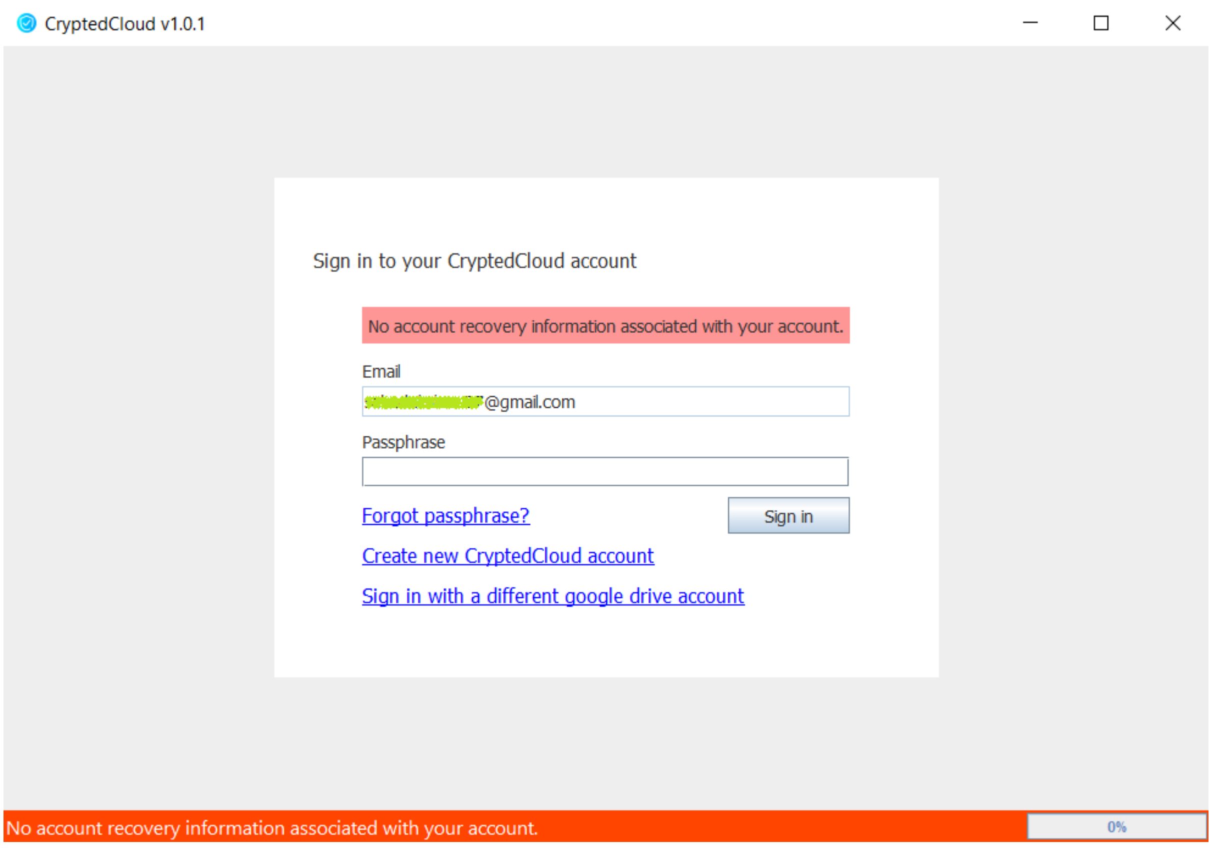
Sign in instead

Finish

Passphrase must be atleast 8 characters long and both 'New passphrase' and 'Re-passphrase' must match.

0%

Figure 4.32: Invalid passphrase provided


CryptedCloud v1.0.1

Sign in to your CryptedCloud account

No account recovery information associated with your account.

Email

[redacted]@gmail.com

Passphrase

Forgot passphrase?

Sign in

Create new CryptedCloud account

Sign in with a different google drive account

No account recovery information associated with your account.

0%

Figure 4.33: Error occurred as no recovery information found for the account

4.2 Server Application

The server side application of our implementation is called “CryptedCloud Server”. The primary objective of our server side is to store data and serve them to the client when requested. As there will be no processing in the server side, we wanted to make an API which can be called from different client applications. We had several different options to choose from e.g. ASP .NET MVC, ASP .NET Core, Spring MVC, Spring Boot etc and all of them were good options.

4.2.1 Technologies Used

As we have implemented our client application in Java, we preferred Java for our server side application too. This will allow us to run our application on any server regardless of the operating system. So we chose Spring Boot framework. We wanted to build our service as a RESTful web service and Spring Boot provides us in-built support for REST services.

For data storage, we needed to have a database. A handful of options were available in this case as well. Initially we planned to use NoSQL database. But due to enhanced security of SQL databases over NoSQL, we chose MySQL database. It was free to use and most importantly it has great community support. Now, database could be accessed in different manners e.g. using raw SQL queries through JDBC library, Object-Relational Mapping (ORM) etc. For flexibility of our development, we used ORM which automatically maps our model classes to the MySQL table. For ORM, we used Java Persistence API (JPA) and Hibernate framework.

4.2.2 Design Principles

While designing the CryptedCloud Server, we had to concentrate on two separate elements, the source code design pattern and the database design.

We tried to follow proper design pattern for our source code. There were several reasons behind that such as, the source code will be more organized, extensions can be made to the server capabilities in the future and it would be much easier for developers to contribute as well. So we chose to follow the Model View Controller (MVC) design pattern. This allowed us to separate business logic from client request handling. Other than that, another important task performed by our server is to communicate with the database. Database communication part is separated from MVC to maintain the separation of concern.

The database design was another crucial decision we had to make as it has serious performance impact if designed improperly. We could only implement ORM through JPA after having a proper schema design. In Figure 4.34, the schema diagram of our relational database is shown.

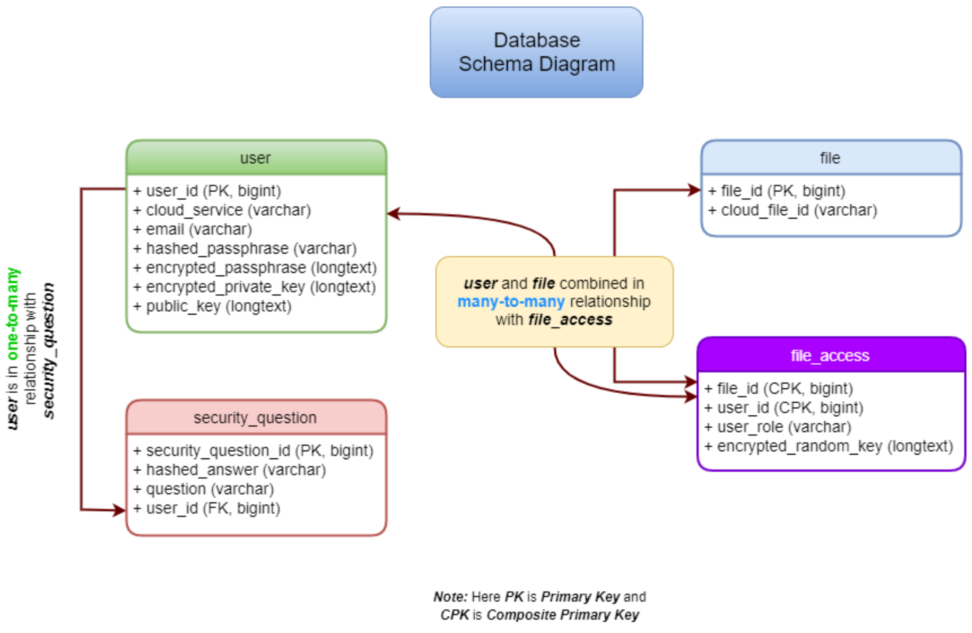

Database Schema Diagram
user
+ user_id (PK, bigint)
+ cloud_service (varchar)
+ email (varchar)
+ hashed_passphrase (varchar)
+ encrypted_passphrase (longtext)
+ encrypted_private_key (longtext)
+ public_key (longtext)
security_question
+ security_question_id (PK, bigint)
+ hashed_answer (varchar)
+ question (varchar)
+ user_id (FK, bigint)
file
+ file_id (PK, bigint)
+ cloud_file_id (varchar)
file_access
+ file_id (CPK, bigint)
+ user_id (CPK, bigint)
+ user_role (varchar)
+ encrypted_random_key (longtext)
user and file combined in many-to-many relationship with file_access
user is in one-to-many relationship with security_question
Note: Here PK is Primary Key and CPK is Composite Primary Key

Figure 4.34: Database schema diagram

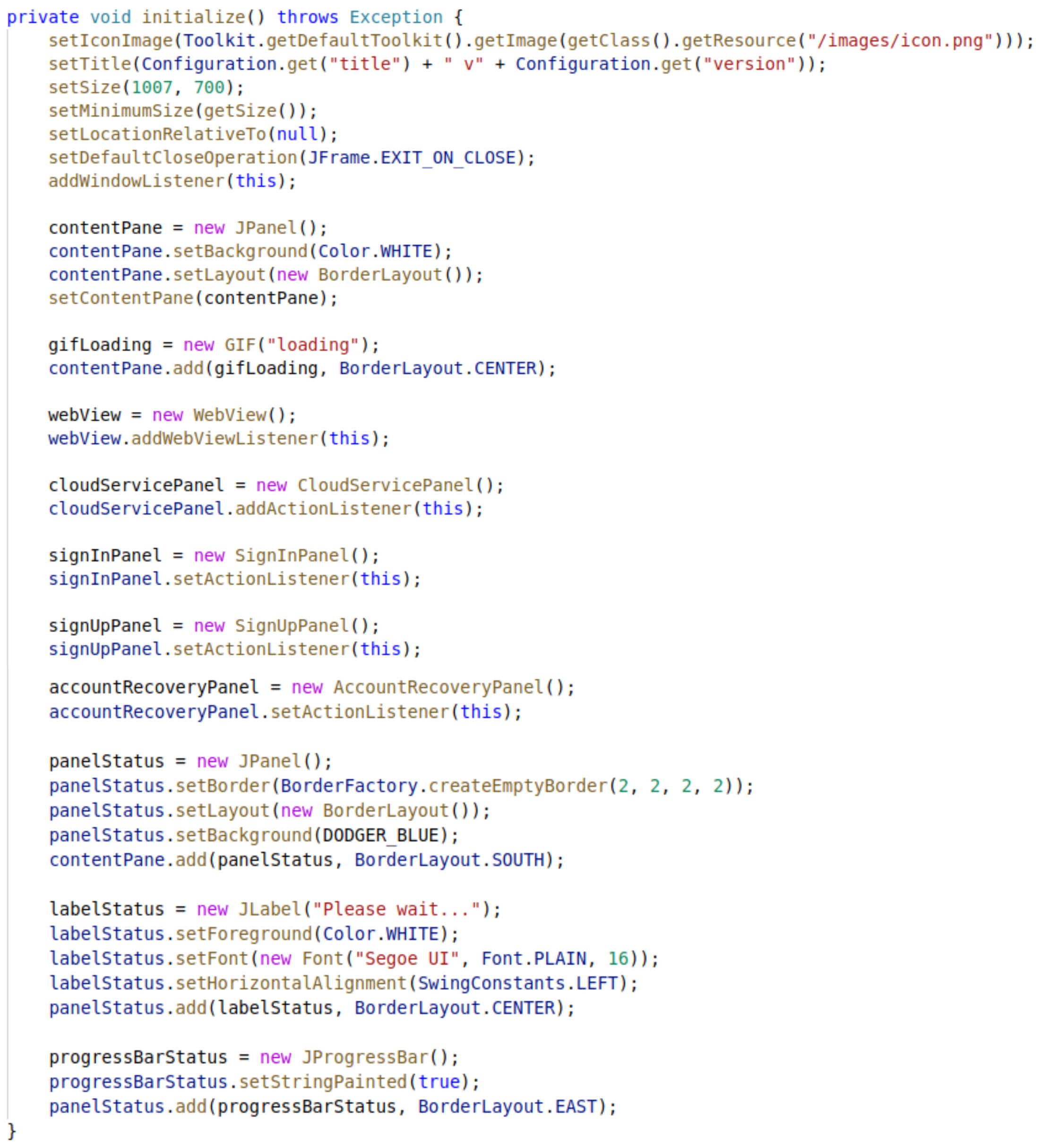
We have four tables in our database, user, security_question, file and file_access. The “user” table contains all the must-have information of a user. The “security_question” table contains information regarding a user’s recovery information. The “user” table has one-to-many relationship with “security_question” table. The “file” table contains information that a file must have and “file_access” table contains access information of a file. It maps a user to his/her files using user_id from “user” table and “file_id” from file table by combining them into a composite key which uniquely identifies an entry of the “file_access” table.

4.3 Internals and Code Snippets

The entire source code of this project is hosted in GitHub as open source (under MIT License) [25]. It is organized in several different packages. e.g. code related to core components are placed in a separate package, code related to UI are kept in another package and so on. As the complete source code is too large to be a part of this report, we will be providing some of the important pieces of the code.

The main window of our client application is initialized in the initialize() method of Frame.java class. All the UI events are also handled in this class. The code snippet below shows the initialize method.

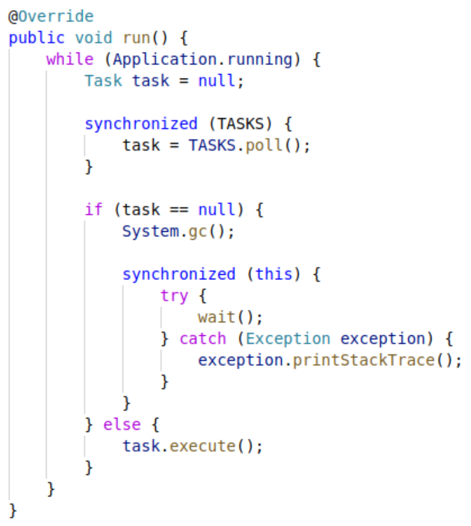
Frame.java



```java
private void initialize() throws Exception {
    setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("/images/icon.png")));
    setTitle(Configuration.get("title") + " v" + Configuration.get("version"));
    setSize(1007, 700);
    setMinimumSize(getSize());
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    addWindowListener(this);

    contentPane = new JPanel();
    contentPane.setBackground(Color.WHITE);
    contentPane.setLayout(new BorderLayout());
    setContentPane(contentPane);

    gifLoading = new GIF("loading");
    contentPane.add(gifLoading, BorderLayout.CENTER);

    webView = new WebView();
    webView.addWebViewListener(this);

    cloudServicePanel = new CloudServicePanel();
    cloudServicePanel.addActionListener(this);

    signInPanel = new SignInPanel();
    signInPanel.setActionListener(this);

    signUpPanel = new SignUpPanel();
    signUpPanel.setActionListener(this);

    accountRecoveryPanel = new AccountRecoveryPanel();
    accountRecoveryPanel.setActionListener(this);

    panelStatus = new JPanel();
    panelStatus.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
    panelStatus.setLayout(new BorderLayout());
    panelStatus.setBackground(DODGER_BLUE);
    contentPane.add(panelStatus, BorderLayout.SOUTH);

    labelStatus = new JLabel("Please wait...");
    labelStatus.setForeground(Color.WHITE);
    labelStatus.setFont(new Font("Segoe UI", Font.PLAIN, 16));
    labelStatus.setHorizontalAlignment(SwingConstants.LEFT);
    panelStatus.add(labelStatus, BorderLayout.CENTER);

    progressBarStatus = new JProgressBar();
    progressBarStatus.setStringPainted(true);
    panelStatus.add(progressBarStatus, BorderLayout.EAST);
}
```

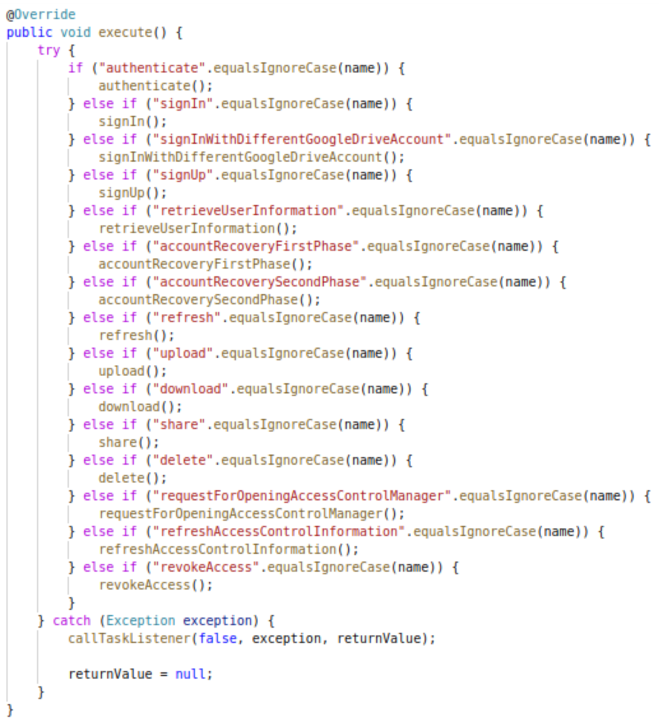
The core components of our client application consists of many different things e.g. background task handling, communicating with server application etc. For execution of the background tasks, we have a task executor. It is started when the application starts. The task executor is defined in TaskExecutor.java class which is inherited from Java's built in Thread.java class. Background tasks are defined in BackgroundTask.java class. TaskExecutor class basically executes the execute() method of BackgroundTask.java class.

TaskExecutor.java



```
@Override
public void run() {
    while (Application.running) {
        Task task = null;

        synchronized (TASKS) {
            task = TASKS.poll();
        }

        if (task == null) {
            System.gc();

            synchronized (this) {
                try {
                    wait();
                } catch (Exception exception) {
                    exception.printStackTrace();
                }
            }
        } else {
            task.execute();
        }
    }
}
```

BackgroundTask.java



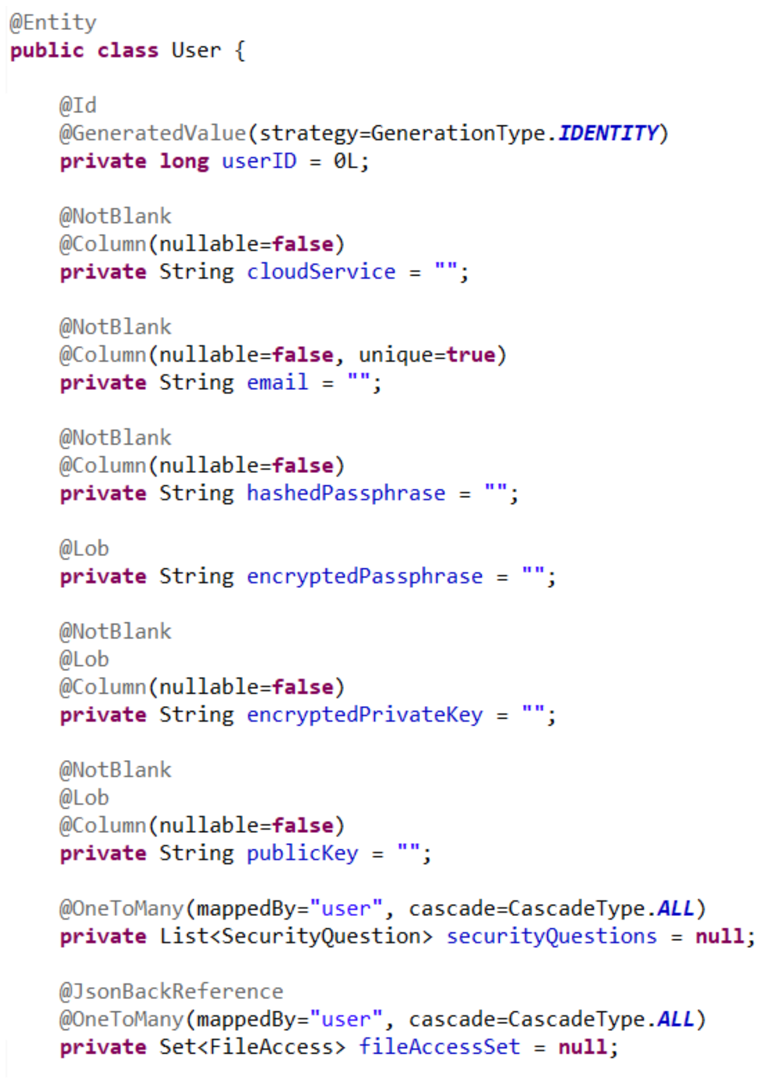
```
@Override
public void execute() {
    try {
        if ("authenticate".equalsIgnoreCase(name)) {
            authenticate();
        } else if ("signIn".equalsIgnoreCase(name)) {
            signIn();
        } else if ("signInWithDifferentGoogleDriveAccount".equalsIgnoreCase(name)) {
            signInWithDifferentGoogleDriveAccount();
        } else if ("signUp".equalsIgnoreCase(name)) {
            signUp();
        } else if ("retrieveUserInformation".equalsIgnoreCase(name)) {
            retrieveUserInformation();
        } else if ("accountRecoveryFirstPhase".equalsIgnoreCase(name)) {
            accountRecoveryFirstPhase();
        } else if ("accountRecoverySecondPhase".equalsIgnoreCase(name)) {
            accountRecoverySecondPhase();
        } else if ("refresh".equalsIgnoreCase(name)) {
            refresh();
        } else if ("upload".equalsIgnoreCase(name)) {
            upload();
        } else if ("download".equalsIgnoreCase(name)) {
            download();
        } else if ("share".equalsIgnoreCase(name)) {
            share();
        } else if ("delete".equalsIgnoreCase(name)) {
            delete();
        } else if ("requestForOpeningAccessControlManager".equalsIgnoreCase(name)) {
            requestForOpeningAccessControlManager();
        } else if ("refreshAccessControlInformation".equalsIgnoreCase(name)) {
            refreshAccessControlInformation();
        } else if ("revokeAccess".equalsIgnoreCase(name)) {
            revokeAccess();
        }
    } catch (Exception exception) {
        callTaskListener(false, exception, returnValue);

        returnValue = null;
    }
}
```

All the operations related to Google Drive can be found in GoogleDriveService.java class. And all the service related calls to CryptedCloud Server is placed in CryptedCloudService.java class. Both the classes are placed under com.cloud.crypted.client.core.services package.

Implementation of AES, RSA and BCrypt can be found on AES.java, RSA.java and BCrypt.java [24] classes respectively under com.cloud.crypted.client.core.cryptography package.

As our server side is built on Spring Boot framework, Model View Controller (MVC) design pattern is followed. Model is the heart of this pattern. It is the main structure or skeleton which manages the data, logic and rules. We have several different models defined in this application. e.g. User model, File model, File Access etc models which are defined in User.java, File.java and FileAccess.java classes respectively. As we are using Object-Relational Mapping (ORM), the model classes are directly mapped to the database.

User.java

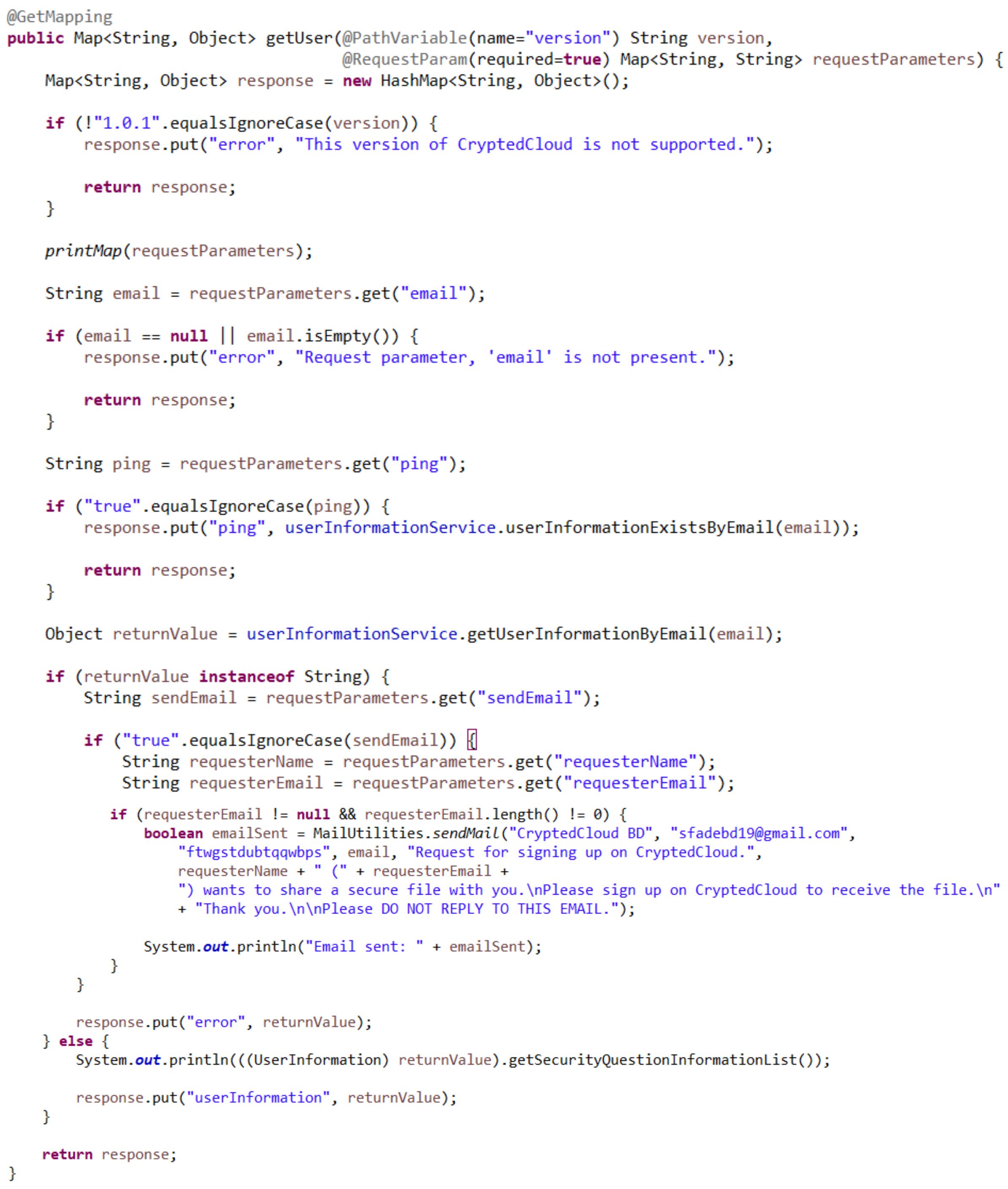


```java
@Entity
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long userID = 0L;

    @NotBlank
    @Column(nullable=false)
    private String cloudService = "";

    @NotBlank
    @Column(nullable=false, unique=true)
    private String email = "";

    @NotBlank
    @Column(nullable=false)
    private String hashedPassphrase = "";

    @Lob
    private String encryptedPassphrase = "";

    @NotBlank
    @Lob
    @Column(nullable=false)
    private String encryptedPrivateKey = "";

    @NotBlank
    @Lob
    @Column(nullable=false)
    private String publicKey = "";

    @OneToMany(mappedBy="user", cascade=CascadeType.ALL)
    private List<SecurityQuestion> securityQuestions = null;

    @JsonBackReference
    @OneToMany(mappedBy="user", cascade=CascadeType.ALL)
    private Set<FileAccess> fileAccessSet = null;
```

Code snippet shown above is not complete User.java class. It only shows the properties and attributes of the class. According to this class, a table named “user” is created in the database by hibernate framework. The columns in the table are defined according to the type and name of the attributes (variables) in this class except for “securityQuestions” and “fileAccessSet”. e.g. the class attribute “userID” is of type long and corresponding column in the database is named “userid” (all lowercase) having the type defined as BigInteger (BIGINT in MySQL). The annotations used on top of the attributes are used to define them more precisely to hibernate framework. Both security question and file access has OneToMany relationship with a user. To define that, @OneToMany annotation is used for both the attributes.

Requests made to the server regarding any of the models are handled by their corresponding model class. e.g. Any request made to the server for retrieving, inserting, updating or even deleting user is handled in user controller. It is defined in UserController.java class.

UserController.java



```
@GetMapping
public Map<String, Object> getUser(@PathVariable(name="version") String version,
                                   @RequestParam(required=true) Map<String, String> requestParameters) {
    Map<String, Object> response = new HashMap<String, Object>();

    if (!"1.0.1".equalsIgnoreCase(version)) {
        response.put("error", "This version of CryptedCloud is not supported.");

        return response;
    }

    printMap(requestParameters);

    String email = requestParameters.get("email");

    if (email == null || email.isEmpty()) {
        response.put("error", "Request parameter, 'email' is not present.");

        return response;
    }

    String ping = requestParameters.get("ping");

    if ("true".equalsIgnoreCase(ping)) {
        response.put("ping", userInformationService.userInformationExistsByEmail(email));

        return response;
    }

    Object returnValue = userInformationService.getUserInformationByEmail(email);

    if (returnValue instanceof String) {
        String sendEmail = requestParameters.get("sendEmail");

        if ("true".equalsIgnoreCase(sendEmail)) {
            String requesterName = requestParameters.get("requesterName");
            String requesterEmail = requestParameters.get("requesterEmail");

            if (requesterEmail != null && requesterEmail.length() != 0) {
                boolean emailSent = MailUtilities.sendMail("CryptedCloud BD", "sfadebd19@gmail.com",
                    "ftwgstdubtqqwbps", email, "Request for signing up on CryptedCloud.",
                    requesterName + " (" + requesterEmail +
                    ") wants to share a secure file with you.\nPlease sign up on CryptedCloud to receive the file.\n"
                    + "Thank you.\n\nPlease DO NOT REPLY TO THIS EMAIL.");

                System.out.println("Email sent: " + emailSent);
            }
        }

        response.put("error", returnValue);
    } else {
        System.out.println(((UserInformation) returnValue).getSecurityQuestionInformationList());

        response.put("userInformation", returnValue);
    }

    return response;
}
```

The code snippet shows the getUser() method. Whenever a request is made to the server for retrieving any information related to user, the request is redirected to this method. More simply, when HTTP GET request is made to fetch user information, this method handles that request.

Other model and controller classes can be found in the similar manner. As our client side is totally detached from the server side, we do not have any view class on our server side.

5 Limitations

Every good thing comes with some limitations. Everything good cannot be packed altogether. e.g. when we solve a problem through dynamic programming, we always have to make a choice between time complexity (increased running time) and space complexity (increased memory usage). If we want performance, we will write our algorithm in such a way that more memory can be utilized. This increases performance but eventually we get more space complexity. On the other hand, if we want to reduce the memory usage, time complexity gets increased. So there is always a tradeoff. Just like everything else out there, our proposed model is not above limitations.

As our proposed model adds an additional layer on top of cloud storage service, it adds a risk of becoming a second point of failure. Suppose a user is using Google Drive as cloud storage and using our model on top of it. If the user uploads a file through our service, the file will be uploaded to the cloud in an encrypted form. Now if he/she wants to download the file and for some reason our server side crashes, the user will not be able to use the file for the time being until the server is up and running again. And this is true for FADE [5] and SFADE+ [7] as well.

Another shortcoming of using our model is reduced performance due to cryptographic operations. We have optimized our design for better performance. But this does not eliminate the cost of cryptographic operations that are performed underneath.

Although our server does not require to be a trusted entity, the contents stored in the database may be altered by the server admin or an intruder through direct access to the database. Suppose an intruder changes the encrypted passphrase of a user by accessing the database. There will not be any security breach but user will surely lose access to the file.

6 Future Work

This project has tremendous improvement opportunity in both architectural design as well as the implementation. The goal of this project was to make sharing functionality less resource consuming while keeping assure file deletion intact. As this project is hosted in GitHub as open source (under MIT License), improvement can be done through community support.

Architectural Improvements

In our proposed model, file sharing requires users to register. We can try to improve the design of our file sharing mechanism so that a user (registered in our CryptedCloud service) can share file(s) with another user without registration. It can be a great scope for research. It will not only improve the design of our file sharing system, but it will also be a great achievement for cryptography.

We only have two roles defined for users in terms of file access control, “owner” and “writer”. The person who has uploaded the file is defined as “owner” and the person with whom the file was shared is defined as “writer”. Adding more roles e.g. “organizer”, “reader” etc. will provide users more flexible file sharing experience.

Furthermore, in our current implementation, when a file is shared with another user, that file cannot be removed by anyone except the owner. So this could be another scope of improvement in terms of file sharing architecture.

Implementation Improvements

Our implementation, CryptedCloud Client can be improved in numerous ways to make it more convenient for day-to-day use. Improved user interface, more efficient use of parallel processing and addition of some useful features can make it more efficient to use.

During the design phase of our client application, our primary focus was to design an elegant looking user interface which looks simple and familiar to use for regular users. We put a lot of effort to improve the user experience. But still, some very common features remained missing. Our client application does not support multiple operations at the same time. e.g. uploading a file and in the meantime downloading two more files is not supported yet. It can be improved easily by applying parallel processing. Drag-n-drop is a common feature now-a-days. But currently it is not implemented in the file explorer of our client application.

As our client application performs all the operations in background, user might want to stop or kill a running task. Adding a task manager could improve the user experience at a greater extent. This would allow users to have complete control over the running tasks.

As we have already mentioned, the implementation part is done in Java which makes it cross-platform among Windows, Linux, macOS and some other desktop operating systems that support Java. But now-a-days, there is an enormous increase in the usage of mobile devices running Android or iOS. People find it more convenient to use mobile apps than desktop applications. So porting the client application to mobile devices will bring this service closer to the users who are concerned about their data privacy.

Performance of the CryptedCloud Server has lots of improvement areas. We have used Object-Relational Mapping (ORM) for schema design of the database. ORM makes it easy to handle the transactions without the need of writing queries manually. But the biggest disadvantage of this approach is lack of performance. Raw queries can be used to improve the performance of database transactions.

Another scope of improvement is to make the server side more scalable. It will allow us to modify our server on the go. Otherwise, even for some minor changes, the source code will have to be modified and redeployed. As ours is a REST server which communicates with clients in JSON format, we followed Data Transfer Object (DTO) design pattern. Though it helped us in managing the data, but it made our server side application less scalable. To make our server more scalable, instead of using DTO, we can use Map data structure (e.g. Linked-Hash Map, Hash Map, Tree Map etc.).

Just like we did for our client application, adding external configuration file will be beneficial for making our server application more modular. Configuration for new client applications can be added without having to restart the server or changing the source code.

Last but not least, for performing cryptographic operations, rather than using Central Processing Unit (CPU), if we could utilize the dedicated Graphics Processing Unit (GPU) whenever available, the application performance would have improved significantly.

Cloud Storage Service with End-to-End Encryption

Currently our model can work flawlessly on top of any of the existing cloud storage services e.g. Amazon AWS, Dropbox etc. For our implementation, we chose Google Drive. But it would be an amazing experience for the users if we could provide a cloud storage service designed with end-to-end encryption in mind.

7 Conclusion

In the era of public network, maintaining privacy is a challenging task. With the increased popularity of cloud storage services, risk of cyber crimes e.g. hacking, data stealing etc. has increased at an alarming rate as technology has advanced tremendously. In this scenario, assured deletion of files has become necessary. Many different methodologies were proposed to mitigate these issues by ensuring privacy and security for the files stored in the cloud storage. But all of them had some disadvantages or flaws that lead us to reimagine the overlay system.

We wanted to have a high performant, modular, future proof and at the same time user-friendly architecture which also resolves the issues that were introduced with all the other models that were proposed. So in this report, we have proposed a new system which concentrates on some key areas such as; reduced resource consumption, secure and faster sharing of files among users, can be used without the need of any support or modification from the cloud service provider and keeping the architecture as simple as possible which makes our system easier to use by users of any caliber.

There are some limitations of our system. e.g. we did not take integrity into consideration while designing the model, as it was assumed that integrity will be provided by the cloud service provider. Again, our proposed model might become the second point of failure as it depends on an additional server. Though we have successfully diminished the necessity of server side being trusted, admin or intruder can change the database contents by accessing the database directly. It will cause security flaw but user will not be able to access his/her valuable data afterwards.

We have some future plans to improve our model by eliminating the limitations and adding new features and performance improvements to our system. We are planning to add more user roles to make our file sharing system more versatile. To improve the user experience, we will allow our client application perform multiple operations concurrently. Some very common features, such as Drag-n-Drop will also be implemented. We also have a plan to bring our client application to mobile devices to serve a large number of users.

To ensure that the improvements are done faster, easier and better, we have published the implementation of our model, CryptedCloud as an open source project. This will allow developers across the globe to contribute and help us improve our implementation. Any architectural changes or modifications can be proposed by any developer and it can also be reflected instantly. Furthermore, if needed, anyone can modify the source code according to his/her requirements.

Our proposed architecture has tremendous scope for research and improvement. If we could improve the design of our file sharing mechanism in such a way that the file(s) can be shared without the need of user registration, it will be a triumph for the entire cryptography.

References

[1] Yamasaki, Y., & Aritsugi, M. (2015). A Case Study of IaaS and SaaS in a Public Cloud. 2015 IEEE International Conference on Cloud Engineering.
[2] M. KAVIS, Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS). New Jersey: John Wiley & Sons, 2014, 51 p.
[3] Cohen, B. (2013). PaaS: New Opportunities for Cloud Application Development. Computer, 46(9), 97–100.
[4] Kandukuri, B. R., V., R. P., & Rakshit, A. (2009). Cloud Security Issues. 2009 IEEE International Conference on Services Computing.
[5] Tang Y., Lee P.P.C., Lui J.C.S., Perlman R. (2010) FADE: Secure Overlay Cloud Storage with File Assured Deletion. In: Jajodia S., Zhou J. (eds) Security and Privacy in Communication Networks. SecureComm 2010. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 50. Springer, Berlin, Heidelberg
[6] Habib, A. B., Khanam, T., & Palit, R. (2013). Simplified File Assured Deletion (SFADE) - A user friendly overlay approach for data security in cloud storage system. 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI).
[7] Nusrat, R., & Palit, R. (2017). Simplified FADE with sharing feature (SFADE+): An overlay approach for cloud storage system. 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC).
[8] Aditya Kaushal Ranjan, Vijay Kumar, Muzzammil Hussain, "Security Analysis of Cloud Storage with Access Control and File Assured Deletion (FADE)", Advances in Computing and Communication Engineering (ICACCE) 2015 Second International Conference on, pp. 453-458, 2015.
[9] Zhang, Y., J. Xiong, X. Li, B. Jin, S. Li, and X. A. Wang. 2016. A multi-replica associated deleting scheme in cloud. In 2016 10th International Conference on complex, Intelligent, and software intensive systems (CISIS). 444–8. doi:10.1109/CISIS.2016.68.
[10] Kamara, S., & Lauter, K. (2010). Cryptographic Cloud Storage. Lecture Notes in Computer Science, 136–149.
[11] Bentajer, A., Hedabou, M., Abouelmehdi, K., Igarramen, Z., & El Fezazi, S. (2019). An IBE-based design for assured deletion in cloud storage. Cryptologia, 1–12.
[12] Sathe, S. C., & Dongre, N. M. (2018). Block Level based Data Deduplication and Assured Deletion in Cloud. 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT).
[13] Sule, M.-J., Li, M., Taylor, G., & Onime, C. (2017). Fuzzy logic approach to modelling trust in cloud computing . IET Cyber-Physical Systems: Theory & Applications, 2(2), 84–89.
[14] Shen, J., Zhou, T., He, D., Zhang, Y., Sun, X., & Xiang, Y. (2018). Block Design-based Key Agreement for Group Data Sharing in Cloud Computing. IEEE Transactions on Dependable and Secure Computing, 1–1.
[15] Xiong, J., Z. Yao, J. Ma, X. Liu, and Q. Li. 2013. A secure document self-destruction scheme: An abe approach. In 2013 IEEE 10th International Conference on high performance computing and communications 2013 IEEE International Conference on embedded and ubiquitous computing, Hunan Province, China. 59–64.
[16] Xiong, J., X. Liu, Z. Yao, J. Ma, Q. Li, K. Geng, and P. S. Chen. 2014. A secure data self-destructing scheme in cloud computing. IEEE Transactions on Cloud Computing 2(4): 448–58.

[17] Xiong, J., Z. Yao, J. Ma, F. Li, X. Liu, and Q. Li. 2014. A secure self-destruction scheme for composite documents with attribute based encryption. Acta Electronica Sinica 42(2): 366–76. http://en.cnki.com.cn/Article_en/CJFDTOTAL-DZXU201402024.htm
[18] Ke, C., Huang, Z., & Cheng, X. (2017). Privacy Disclosure Checking Method Applied on Collaboration Interactions Among SaaS Services. IEEE Access, 5, 15080–15092.
[19] Mengistu, T., Alahmadi, A., Albuali, A., Alsenani, Y., & Che, D. (2017). A "No Data Center" Solution to Cloud Computing. 2017 IEEE 10th International Conference on Cloud Computing (CLOUD).
[20] Provos, Niels & Mazieres, David. (1999). A Future-Adaptable Password Scheme.
[21] Introduction to Google Drive API, https://developers.google.com/drive/api/v3/about-sdk
[22] Using OAuth 2.0 to Access Google APIs, https://developers.google.com/identity/protocols/OAuth2
[23] Jackson Project Home @github, https://github.com/FasterXML/jackson
[24] BCrypt hashing algorithm implementation in Java, https://github.com/djmdjm/jBCrypt/blob/master/src/org/mindrot/jbcrypt/BCrypt.java
[25] CryptedCloud Source Code Repository, https://github.com/shahadul-17/CryptedCloud