NTNU
Norwegian University of
Science and Technology

Long-Term Confidential Data Storage by Distributed Secret Shares

Merete Løland Elle

Master of Telematics - Communication Networks and Networked Services
Submission date: June 2017
Supervisor: Stig Frode Mjølsnes, IIK
Co-supervisor: Ruxandra Florentina Olimid, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Long-Term Confidential Data Storage by Distributed Secret Shares

Student: Merete Løland Elle

Problem description:

Long time preservation of data confidentiality is often a requirement in real-life applications like legal documents and health records. On one hand, data must be readily available to qualified parties for many years. On the other hand, the data must be kept confidential from unauthorized parties, even under incidents like hardware or software errors and attacks. Mechanisms of secret sharing represent an alternative to the classical solution that requires both encryption (for data confidentiality) and backup policies (for data availability). Secret sharing based systems provide data secrecy and reliability simultaneously, by splitting data among multiple storage nodes located in different physical locations. Data can be recovered when a qualified subset of nodes are up and running, while unqualified sets of nodes cannot leak any information about the original data. This approach is a possible solution for secure distributed cloud storage where the servers are located with distinct providers.

The student will design, implement, and experiment with secret sharing with distributed storage systems. The student will focus on the design problems of creating a user-friendly Android application for storing and retrieving personal passwords based on mechanisms of distributing secret shares among a set of internetworked servers. By storing passwords in this application, the user should have easy and user-friendly access to the passwords, while not compromising the security.

Responsible professor: Stig Frode Mjølsnes, IIK
Supervisor: Ruxandra-Florentina Olimid, IIK

Abstract

There are several mobile password managers on the marked, where the most popular of these uses the classical solution for storage which requires both encryption and backup policies. If quantum computers become a reality, the security of encryption methods based on factoring primes or doing modular exponentiation is threatened. For threshold secret sharing schemes, an unauthorized set of shares of the secret provides no information about the secret. By this, one can say that secret sharing is information-theoretically secure, which means that it cannot be broken even when an attacker has unlimited computing power.

In this thesis, the development of a password storage mobile application for Android is presented. The mobile application implements secret sharing for confidentiality and uses cloud storage services for storing the shares. A password is divided into three pieces, where two or more are needed to reconstruct the password. Together, this manifests itself as a (2,3) threshold scheme. The cloud storage services implemented are Dropbox, Google Drive, and Microsoft OneDrive.

User tests were conducted for testing the functionality and the User Interface (UI) of the application. The result was a "PASSED" score on 98,2%, which indicated that the functionality performed better than expected and the alternative hypothesis H_1 was supported. The feedback from the test subjects stated that the application looked good and worked well, but some of the solutions could have been optimized in regards to the UI.

Sammendrag

Det finnes flere mobilapplikasjoner for passordadministrering på markedet, hvor de mest populære av disse benytter seg av den klassiske løsningen for lagring, som krever både kryptering og retningslinjer for reserveløsninger. Hvis kvantedatamaskiner blir en realitet, vil sikkerheten til krypteringsmetoder basert på faktorisering av primtall og modulær eksponering være truet. For det som er kjent som *"threshold secret sharing schemes"*, vil et uautorisert sett med deler av en "hemmelighet" ikke gi noe informasjon om hemmeligheten. På bakgrunn av dette kan man si at denne teknikken er "information-theoretically secure", noe som innebærer at selv ubegrenset med databehandlingskraft ikke vil kunne gi en angriper informasjon om den såkalte hemmeligheten.

I denne oppgaven er utviklingen av en Android mobilapplikasjon for lagring av passord presentert. Mobilapplikasjonen implementerer secret sharing for konfidensialitet, og bruker skylagringstjenester for å lagre de såkalte delene. Et passord blir delt inn i tre deler, hvor to eller flere av disse kreves for å få tak i passordet. Dette manifesterer seg som et "(2,3) threshold scheme". Skylagringstjenestene som er implementert er Dropbox, Google Drive og Microsoft OneDrive.

Brukertester ble gjennomført for å teste funksjonaliteten og brukergrensesnittet til applikasjonen. Resultatet ga en bestått-score på 98,2%, noe som indikerte at funksjonaliteten fungerte bedre enn forventet og gir tilstrekkelig bevis for at den alternative hypotesen H_1 er sann. Tilbakemeldingene fra testobjektene inneholdt kommentarer om at applikasjonen så bra ut og fungerte slik den skulle, men at noen av løsningene kunne ha blitt optimalisert med tanke på brukergrensesnittet.

Preface

This Master's thesis is written as a part of the Master of Science programme in Telematics - Communication Networks and Networked Services at the Norwegian University of Science and Technology (NTNU), spring 2017.

I would like to thank my supervisors Professor Stig Frode Mjølsnes and Ruxandra-Florentina Olimid for their support during the work with the master project, as well as for the valuable comments and suggestions. I would also like to thank my focus group for providing me with helpful and relevant feedback.

The code for the application is not included in the appendices because of the amount of pages required, but uploaded to a repository on GitHub. It can be found by using the following URL: `https://github.com/meretele/SecretSharing`.

Trondheim, June 2017

Merete Løland Elle

Contents

List of Figures

List of Tables

List of Listings

Chapter 1 Introduction

This chapter introduces the thesis as a whole. It presents the background and motivation for the thesis and gives an introduction to the concept of secret sharing. The goal and research question for the topic is presented, as well as the methodology used for the field of study. It also includes the contributions, related work and lastly the outline of the thesis.

This thesis is not associated with or sponsored by Google, Microsoft or Dropbox, Inc. The thesis does not consider the security of these external services themselves, but takes into consideration the possibility of one them being compromised, by using secret sharing for confidentiality. Throughout the thesis, term *clouds* will be frequently used when referring to the cloud storage services.

1.1 Background and Motivation

Today's Internet services rely heavily on text-based passwords for user authentication, and it is a common understanding that one should never give away passwords to anyone, or at least not to anyone whom one do not trust completely. Still, according to a recent survey of 276 IT professionals commissioned by Sungard Availability Services, 59% cited employee password sharing as their main security concern [Sun]. Additionally, a study done by researchers at Dartmouth College, the University of Pennsylvania and the University of Southern California shows how healthcare employees circumvent security rules [KSBK15]. They found that workarounds to cyber security are the norm, rather than the exception, and in several hospitals, passwords are written down everywhere - "no one wanted to prevent a clinician from obtaining emergency supplies because they didn't remember the code" [KSBK15].

Once a password gets written down, it becomes available to the wrong pair of eyes. One may argue that convenience trumps the possibility of the password being stolen, but is this the reality when the password can be used to access sensitive data? What if something as available and convenient as a mobile phone could be used for securely storing passwords?

There are several mobile password managers, the most popular of these uses the classical solution for storage which requires both encryption and backup policies [Fit16]. If quantum computers become a reality in the future, the security of encryption methods based on factoring primes or doing modular exponentiation are threatened [Nor]. For threshold secret sharing schemes, an unauthorized set of shares of the secret provides no information about the secret. By this, one can say that secret sharing is information-theoretically secure, which means that it cannot be broken even when an attacker has unlimited computing power. Does this mean, in a post-quantum scenario, that secret sharing schemes could be a solution for providing security?

1.2 Introduction to Secret Sharing

Secret sharing is a method for dividing a secret into pieces and distribute these among a group of participants. The secret can only be reconstructed when a certain number of these pieces are combined, and the individual pieces give no information about the secret [Sha79].

1.2.1 Secret Sharing Schemes

One practical example of secret sharing is having a vault in a bank that needs to be opened every day. The bank has three employees, but it is not preferable to entrust one person with the vault combination. Hence, the bank wants a system where two of the three employees can open the vault, while an individual can not do so. This can be solved by means of a secret sharing scheme [Sti92].

Secret sharing schemes can have the property of being verifiable. This is called *Verifiable Secret Sharing Schemes (VSS)*, and has the functionality of verifying that a received share is "valid," detect incorrect shares returned by faulty or compromised servers and also to recognize incorrect sharing writes [SB05].

1.2.2 Shamir's Secret Sharing Scheme

In 1979, Shamir introduced the concept of secret sharing in a paper titled *"How to Share a Secret"* [Sha79]. The motivation for this new cryptographic primitive was to enable the construction of robust key management schemes. Placing an encryption key in a single, well-guarded location is unreliable, as a single misfortune could make the key unavailable, and by storing multiple copies of the key at different locations will increase the danger of security branches.

He purposed a technique of dividing data D into n pieces in such a way that D is easily reconstructable from any k or more pieces, but even the complete knowledge of *k-1* pieces reveals absolutely no information about D, as all possible values are equally likely. This is called a *(k, n) threshold scheme*. The reason for using a threshold scheme is that it might be impractical to count on all participants to recover the secret, especially if the number of shares is high. As stated in the paper, "threshold schemes are ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate" [Sha79], which prevents individuals from making important, and supposedly malicious, decisions by themselves.

The scheme is based on polynomial interpolation, which is the method of finding a polynomial which goes through some given points. The essential idea is that it takes k points to define a polynomial of degree *k-1*, which means that to define e.g. a parabola (polynomial of degree two), three points are needed. Figure 1.1 illustrates an example of having two points, which would be sufficient to define a line, but one can draw all the possible parabolas for the given field through these points. This means that knowing two points when defining a parabola is useless, as all parabolas are equally plausible. This is called *perfect secrecy*.

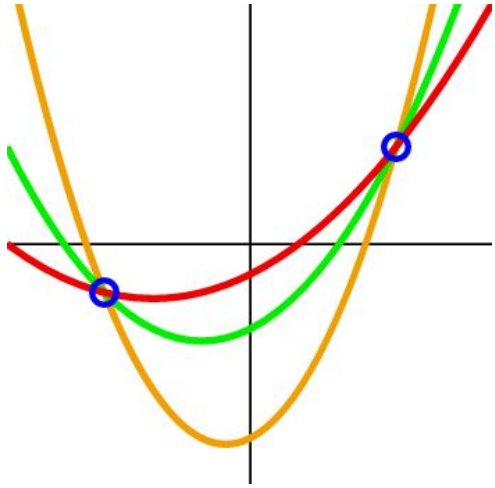

Figure 1.1: Illustration of drawing several parabolas when only two points are given, all parabolas is equally plausible to be the correct one [Vls].

1.3 Goal and Research Question

The goal and research question creates the basis of this thesis by being the focal points of the study.

1.3.1 Goal

The focus of this thesis is to study the possibility of using secret sharing for confidentiality in a mobile application for storing passwords. As presented in Section 1.2, the concept of secret sharing was introduced already in 1979, and one example for applying this technology has been presented as a survivable information storage called the *PASIS architecture* [BWS+00]. It uses threshold schemes to spread information to servers, and *PASIS agents* to communicate with the nodes to read and write - thus hiding decentralization from the users. It claims to provide better confidentiality, availability, durability, and integrity of information than conventional replication.

By using the concept of such a storage system, and implementing it in a mobile application, one could create an application that acts like the conventional password storage managers implementing encryption for confidentiality but provides a storage solution that is information-theoretically secure.

There is at least one mobile application for Android which already implements secret sharing for saving passwords, but it is new and under development, and uses other devices for storing the shares. The overall goal for this thesis is to create a mobile application for storing password, using secret sharing for confidentiality, and cloud storage services for storing the shares.

1.3.2 Research Question

Based on the goal for the thesis, the overall question to answer is: *would it be possible to make an Android application for storing password, that implements secret sharing for confidentiality, and is both user-friendly and secure?*

1.4 Methodology

The research methodology used in the work with this thesis is divided into a design phase, a development phase and a testing phase.

For the design phase, an initial prototype was created to establish an idea of how the UI should appear to the user. This was used as a base when developing the application, and the UI was revised several times. For the development phase, the official Integrated Development Environment (IDE) for Android, Android Studio, was used for developing the application - with Java as the programming language [Goo17b]. Three clouds were connected to the application for storing the shares. For the testing phase, user tests were conducted to evaluate if the application was functioning properly, and also if it met the criteria of being user-friendly.

1.5 Contributions

This thesis shows the whole process of developing an Android application from scratch. The application's purpose is to store passwords, using secret sharing for confidentiality, and cloud storage services for saving the shares. The main contribution of the thesis is the application, as no research has found that an application equal to the one presented has been developed before. The UI has been custom made for this application, and the application itself has been made by combining existing code with self-written code to make the application perform properly. User tests have been created and conducted, and the results are analyzed and explained.

1.6 Related Work

As mentioned in Section 1.1, several mobile password managers exist, but most of these uses encryption for confidentiality [Fit16]. Available in Google Play, there is an application called *Convenient Password Manager* [Cra16]. It is currently unreleased and in alpha testing, and was latest updated on September 26, 2016. Similar to the application in this thesis, the application uses an implementation of Shamir's secret sharing to split up a password. However, instead of storing the shares on clouds, the shares are distributed to the user's devices that have the application installed. By attempting to recover the password on one device, the connected devices will get a recovery request, where the user must accept or decline. By accepting the request on a certain amount of devices, the password is recovered.

1.7 Outline

This thesis is divided into seven chapters, including this introductory chapter. The outline is as follows.

Chapter 2 Presents the background theory about the topics discussed in the thesis, including secret sharing, Android development, cloud storage APIs and graphical user interfaces.

Chapter 3 Describes the user interface of the application. Also included is the initial prototype designed.

Chapter 4 Describes the functionality of the application, by providing diagrams to show the workflow of the system and code snippets to explain the how the application is built.

Chapter 5 Presents the plan, hypothesis, setup and results of the user test for the application.

Chapter 6 Presents the discussion of the work done during the thesis, and explains the choices that have been made regarding the application.

Chapter 7 Presents the conclusion for the thesis and proposals for future work.

Chapter 2
Background Theory

This chapter introduces the background theory regarding Android development, as well as the integration and authentication with the cloud storage services.

2.1 Introduction to Android Development

Android is an open-source mobile Operating System (OS) owned by Google, primarily developed for mobile devices with touchscreens [Gooa]. It has, as for April 2017, approximately 65% of the marked share globally [Net].

The Google-owned IDE, Android Studio, is the official IDE for Android, providing tools for building applications for all Android devices. The following sections will cover the fundamental building blocks used when creating an Android application.

2.2 Core Framework Components

The application components are the building blocks of an Android application, and each component is an entry point through which the system or a user can enter the application [Goo17c]. There are four types of application components, each serving a distinct purpose:

- Services.
- Broadcast receivers.
- Activities.
- Content providers.

Activities, services, and broadcast receivers are activated by an asynchronous message called an *intent*, which binds individual components to each other at runtime [Goo17c]. For activities and services, an intent defines the action to perform, and for broadcast receivers, the intent defines the announcement being broadcast [Goo17c].

2.2.1 Services

A service is a separate part of the application that runs in the background, without providing a UI [Goo17c]. For example, a service might synchronize data in the background while the user is in a different application.

2.2.2 Broadcast receivers

A Broadcast receiver is a part of the application that enables the system to deliver events to the application outside of a regular user flow, allowing the application to respond to system-wide broadcast announcements [Goo17c]. For example, the system may send a broadcast when the device starts charging.

2.2.3 Activities

An activity serves as the entry point for an application's interaction with the user, representing a single screen with a UI [Goo17c]. For example, a note taking application might have one activity showing all the saved notes, and another for composing a new note. An application typically comprises multiple independent activities, where one is specified as the *main activity*, being the first screen to appear when the user launches the application [Goo17j].

The activity instances transitions though different states in their lifecycle [Goo17m]. The activity class provides a core set of six callbacks that allow the activity to know that a state has changed, including *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()*, and *onDestroy()*. A simplified illustration of the activity lifecycle is shown in Figure 2.1.

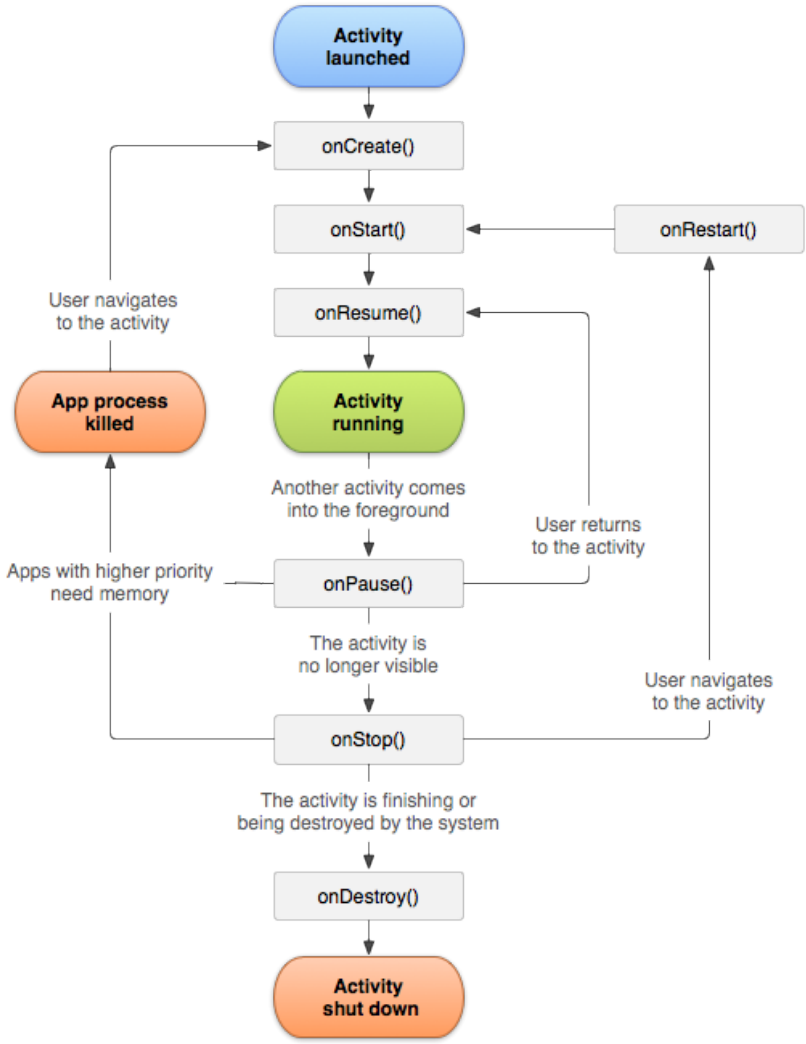

Figure 2.1: A simplified illustration of the activity lifecycle [Goo17m].

2.2.4 Content providers

A content provider manages a shared set of data that can be stored in any persistent storage location that the application can access [Goo17c]. The content provider enables the sharing of data between applications and can be configured to allow other applications to securely access and modify the application data as illustrated in Figure 2.2 [Goo17f].

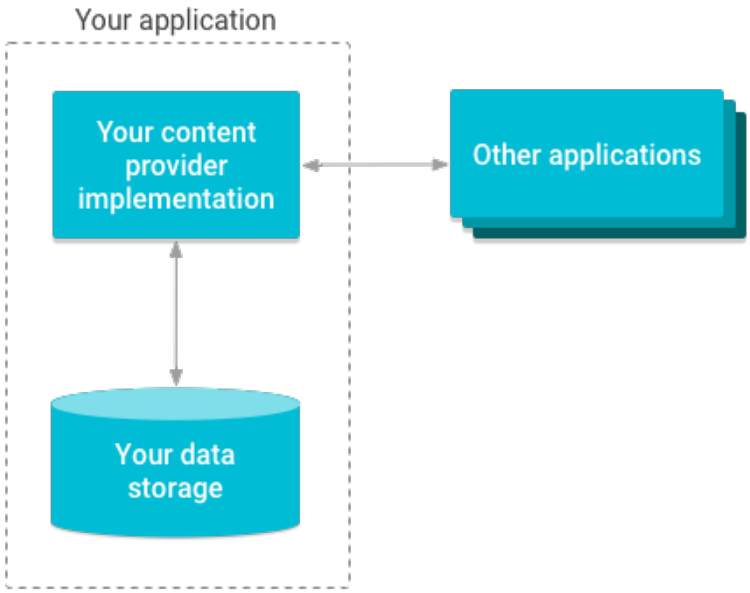

Figure 2.2: An overview of how content providers manage access to storage [Goo17f].

2.3 The Manifest

The *manifest file*, located in the root of the application project directory, is an important part of the application, where all its components must be declared [Goo17c]. It is, in addition to declaring the applications component, used for identifying any user permissions, as well as for declaring the minimum Application Programming Interface (API) level required, hardware and software features used or required, and API libraries that the application needs to be linked against [Goo17c]. An example of a manifest file is shown in Listing 2.1.

Listing 2.1: Example snippet of a *AndroidManifest.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    <application android:icon="@drawable/icon" ... >
        <activity android:name="com.example.project.MainActivity"
                  android:label="@string/app_name" ... >
                   <intent-filter>
                        <action
                            android:name="android.intent.action.MAIN"
                            />
                    </intent-filter>
        </activity>
        ...
    </application>
</manifest>
```

2.4 Security

The security models for Android includes code signing, application isolation, permission model, file system encryption, and generic exploit mitigation protection [Li16].

The purpose of code signing is to prove the identity of an application's author by signing the app with the developer's public key and only allowing updates to be made if the developer has the private key [Li16]. For application isolation, each Android application lives in its own security sandbox, the Android Application Sandbox, which isolates the application data and code execution from other applications [Goo17c]. If the application needs to use resources outside of the sandbox, the application has to request the permission. The permissions are declared by listing them in the manifest file described in Section 2.3 [Goo17g]. Android also supports file system encryption, where the user-created data is encryped before it is saved [Goo17i]. Also, generic exploit mitigation protection is handled by using technologies like safe_iop and ASLR to defend against control hijacking [Li16]. A summary of the security models is shown in Figure 2.3.

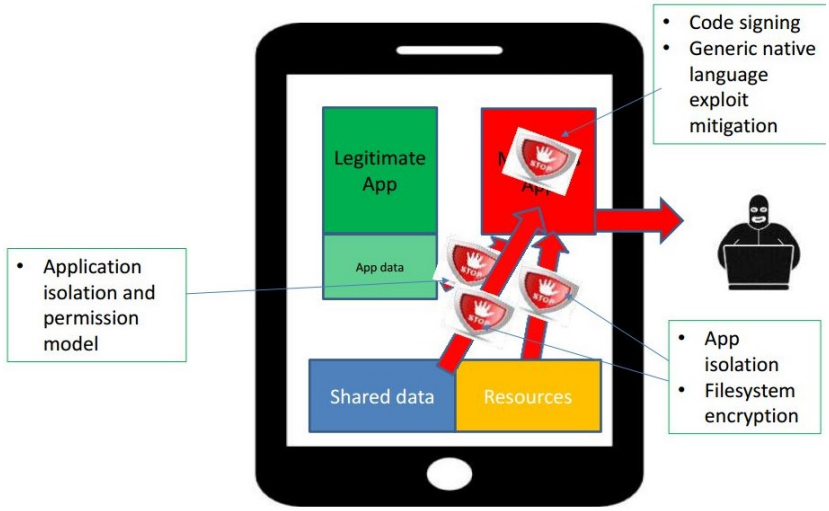

Figure 2.3: Android security model summary [Li16].

2.5 User Interface

As one of the main goals is to make a user-friendly application, the UI is of particular importance. The UI allows for the user to interact with the application through an interface, which in this case is a Graphical User Interface (GUI). When designing the visual composition of a UI, the general goal is to allow for the user to reach their goal efficiently.

One of the pioneers in trying to evaluate the user experience on digital platforms objectively is Jakob Nielsen. He produced ten heuristics, or general principles, for user interface design, which are still valid today and is listed below [Nie95]. These principles have been used as a guideline when designing the application.

1. **Visibility of system status**
   The user should always be kept informed about the system status through feedback within reasonable time.
2. **Match between system and the real world**
   Rather than using system-oriented terms, the system should speak in words, concepts, and phrases familiar to the user. The information should appear in a logical and natural order.
3. **User control and freedom**
   The system should support undo and redo, as users often choose

system functions by mistake. A marked exit from the unwanted state should be available, without having to go through an extended dialogue.

4. **Consistency and standards**
   The system should be consistent, and the user should not be insecure on whether different situations, words or actions mean the same thing.
5. **Error prevention**
   Good error messages are important, but careful design for preventing a problem from occurring is better. One should either eliminate error-prone conditions or check for them and have the user confirm the action before committing.
6. **Recognition rather than recall**
   The user should not have to remember information throughout dialogues. Objects, actions, and options should be made visible, and instructions for usage of the system should be visible or easily retrievable.
7. **Flexibility and efficiency of use**
   Accelerators - unseen by new users - may often speed up the interaction for the expert user, and thus the system can cater to both inexperienced and experienced users. The system should allow users to tailor frequent actions.
8. **Aesthetic and minimalist design**
   Dialogues should not contain irrelevant or unnecessary information, as this extra information competes with the relevant information and reduces its relative visibility.
9. **Help users recognize, diagnose, and recover from errors**
   Error messages should not be explained in codes, but in plain language, with precise indications of the problem, and suggestions on how to solve it.
10. **Help and documentation**
    Providing the user with help and documentation may be necessary. Information should be easy to search through, focused on the user's tasks, contain specific steps to be followed, and not be too big.

2.6 Cloud Storage Services APIs

An API serves as an interface between different software programs for communicating with each other, and a cloud storage API connects a locally-based application to a cloud-based storage system. By this, a user of the application can send data to the cloud and access the data stored in it. Three of the major cloud storage providers on the marked today are Dropbox, Google Drive and Microsoft OneDrive [Cas]. They provide from 2 to 15 GB of free storage, and all cloud storage services encrypts the data in transit [Win].

All of these clouds use the standard OAuth 2.0 authentication scheme for authentication and for generating access tokens [Mic15, Goo17d, Dro17c]. The OAuth 2.0 is an authorization framework that enables applications to obtain limited access to users resources, without the users having to share their log-in credentials with the application [DH12]. The OAuth process will generate access tokens used to uniquely identify both the application and the user.

2.6.1 Dropbox

First, the app has to be registered on the Dropbox Platform and receive an *app key* used to identify the mobile application. Then, after a user has completed the OAuth 2.0 authorization flow, the application receives the access token and uses this to create a Dropbox client. The client, *DbxClientV2*, is used to make remote calls to the Dropbox API user endpoints [Dro17a].

2.6.2 Google Drive

The Google Drive API is a native Android API for accessing Google Drive content [Goo17k]. Similar to Dropbox, an *OAuth 2.0 client ID* must be created for the application to generate an access token. The client, *GoogleApiClient*, is used for the connection between the application and the API [Goo17a].

2.6.3 Microsoft OneDrive

Like in the previous clouds, the application has to be represented by an *application ID* in API calls. Then, after a user has completed the OAuth 2.0 authorization flow, the application receives the access token and the client, *IOneDriveClient*, is used for making calls to OneDrive API [Mic15].

Chapter 3
The User Interface of the *SecretSharing* Application

This chapter presents the initial prototype of the application layout made before creating the application, as well as the final user interface. Screenshots and sketches are included to give an impression of how a user will interact with the application. The term "platform name" corresponds to the ID given to a particular password, and the term "password" is referring to the stored password.

3.1 Initial prototype

Before developing the application, sketches were created to get a rough idea of how the final result should look. The main reason for doing this was to uncover all the base functionality needed and combine it together in a system.

Figure 3.1 shows a splash screen (or a "welcome screen") containing the application logo, the main page where the user would see the status of the cloud connections intuitively, and a draft of how a drop-down menu could be implemented. It shows some cloud providers, where the connection status is visualized by "OK" and "NOT OK" symbols. Also, it shows the overall status, which is this case is "OK" since only one of the four clouds is up - two, or more, would need to be down for the status to be "NOT OK," and the user not being able to retrieve the password. As explained in Chapter 1, Section 1.2, this is called a (3,4) threshold scheme.

Figure 3.2 shows how it was thought that the user could create the passwords. Here, the *PASSWORD-ID* indicates the ID, or platform name, of the password. By simply pressing a button, the user should be able to generate shares from the password and save them to the clouds.

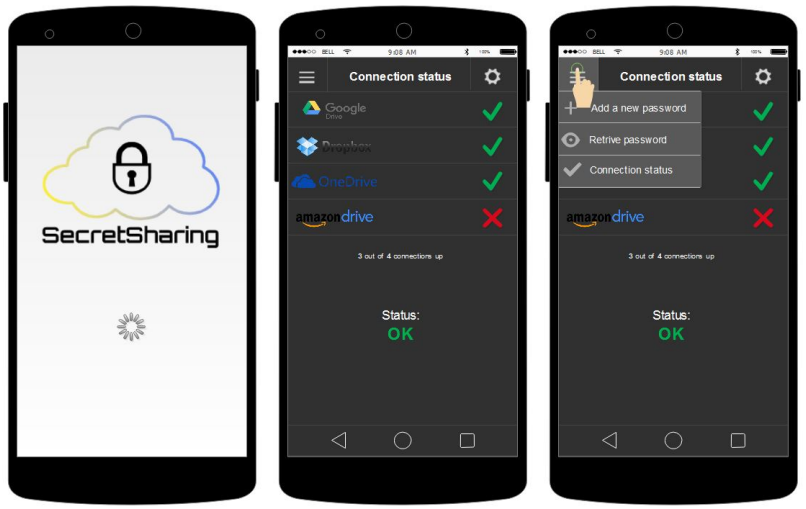

Figure 3.1: Draft of the application; left: the loading screen, middle: connection status page, right: connection status page with a drop-down menu.

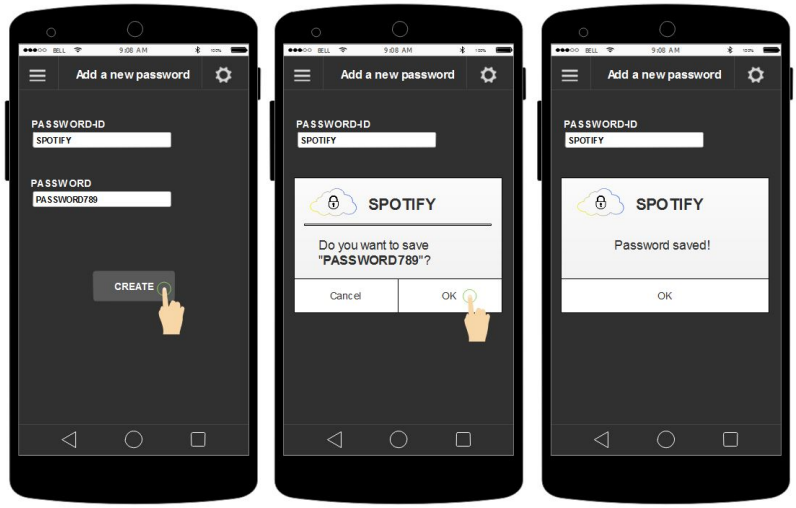

Figure 3.2: Draft of the application; left: creating a new password, middle: pop-up asking if the user wants to save the password, right: the password is saved.

For retrieving the password, the idea was that the user clicks on the one to retrieve and then get the password returned in a pop-up, as shown in Figure 3.3. Here, the user has a selection of the platform names (e.g. "Facebook") already created, which should have their respective shares stored in the clouds.

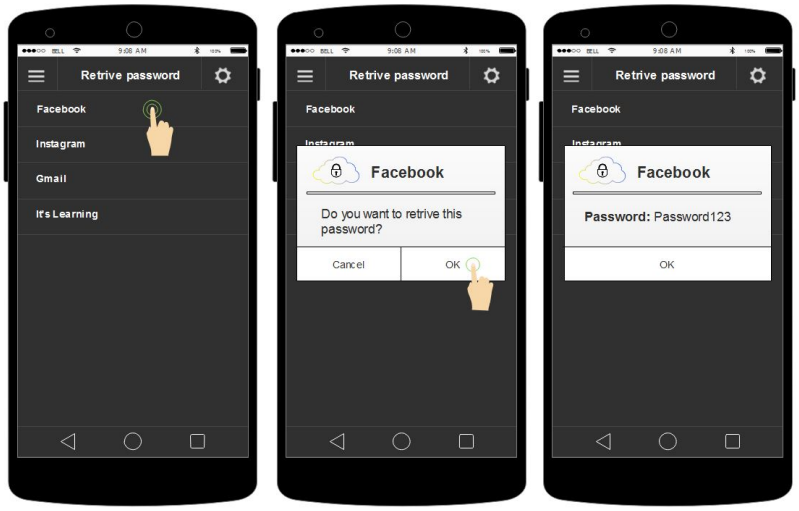

Figure 3.3: Draft of the application; left: clicking on a password on the retrieve passwords page, middle: pop-up asking if the user wants to retrieve the password, right: the password retrieved.

3.2 Final User Interface

The UI has been redesigned several times during the development of the application, as new features were added or removed. The overall color scheme used for the application is black, white and various shades of gray, with a red font used for critical messages. Also, based on the user tests in Chapter 5, minor changes were made to optimize the UI. In the screenshots of the application, red boxes around the small notifications, also known as "toasts," are included to make it easier for the reader to understand which messages to focus on.

3.2.1 Logging In

The first thing the user will see after clicking the icon and launching the application, is a splash screen, as seen in Figure 3.4. A splash screen is commonly known as a "welcome screen," and is shown while the application is loading. The splash screen shows the applications logo, which consists of a cloud, a lock, and the application name.

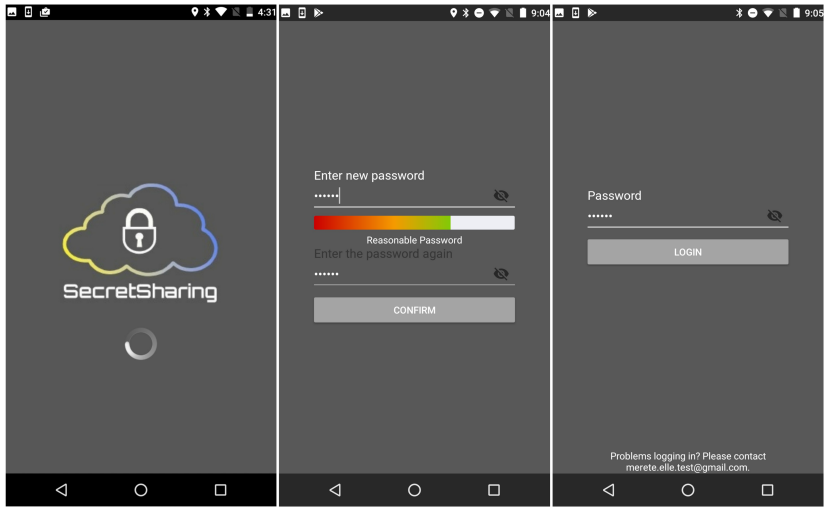

Figure 3.4: Screenshot of the application; left: splash screen, middle: creating a new application password, right: log-in page.

If the user has not set an application password before, a screen for creating a password will be presented. A progress bar, shown in Figure 3.4, will give the user an indication of whether the typed password is weak, reasonable or strong. This evaluation is based on the length of

the password, as well as the number of symbols, digits, uppercase and lowercase letters. The text in the password fields is hidden by default, but the user can display it by pressing the eye icon.

After having typed the same password twice and pressed the confirm button, the user is directed to the log-in page to fill in the password. This log-in page is prompted every time the application is opened, to create an extra barrier for a potential attacker. The text in the password field is hidden by default here as well. As seen in Figure 3.5, the user only has five attempts to submit the right password, without having to re-open the application. When the wrong password is provided, a message will notify the user that the password is not correct. This feature is also implemented to create an extra obstacle for an attacker trying to gain access. By making it harder to run an automated brute-force script and making the attacker re-open the application after five attempts, it may provide the time needed for the user to realize that the phone is lost or stolen and delete the shares manually.

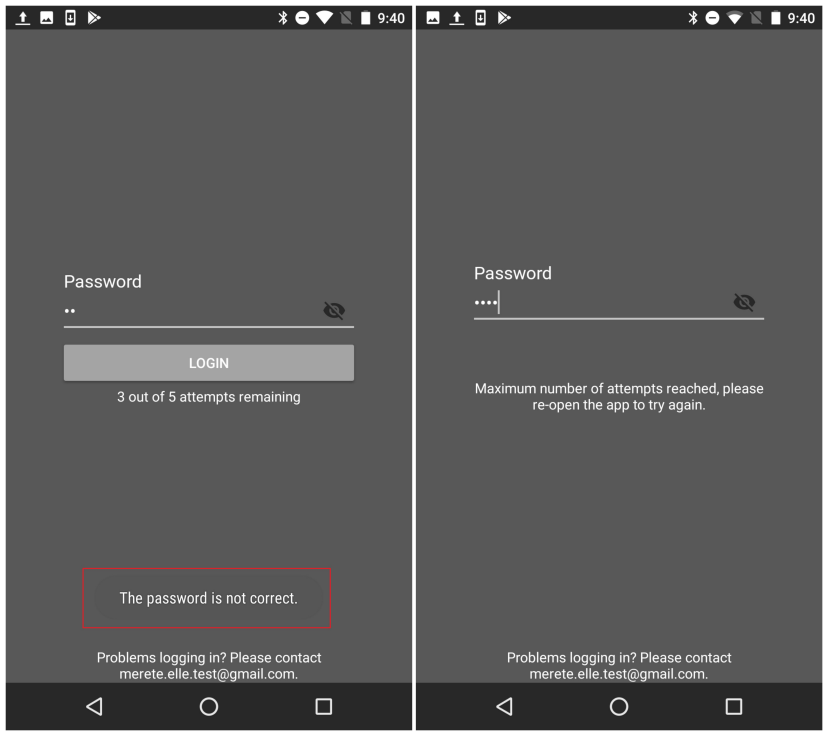

Figure 3.5: Screenshot of the application; left: error message when password is not correct, right: maximum number of attempts reached.

As seen in Figure 3.5, an email address is provided on the bottom of the page for the user to contact if there are any problems while logging in. As the functionality for having lost the application password is not implemented, this is the temporary solution. The solution can also be related to the 10th heuristic mentioned in Chapter 2, Section 2.5.

3.2.2 Connection Status

The connection status page is the main page, or the entry point, of the application. When the user is successfully logged in, this page is loaded. The log-in buttons contain the icons for the clouds, as well as their company name. This usage is done in accordance with the guidelines for branding, presented by the cloud storage services [Dro, Mic, Goob].

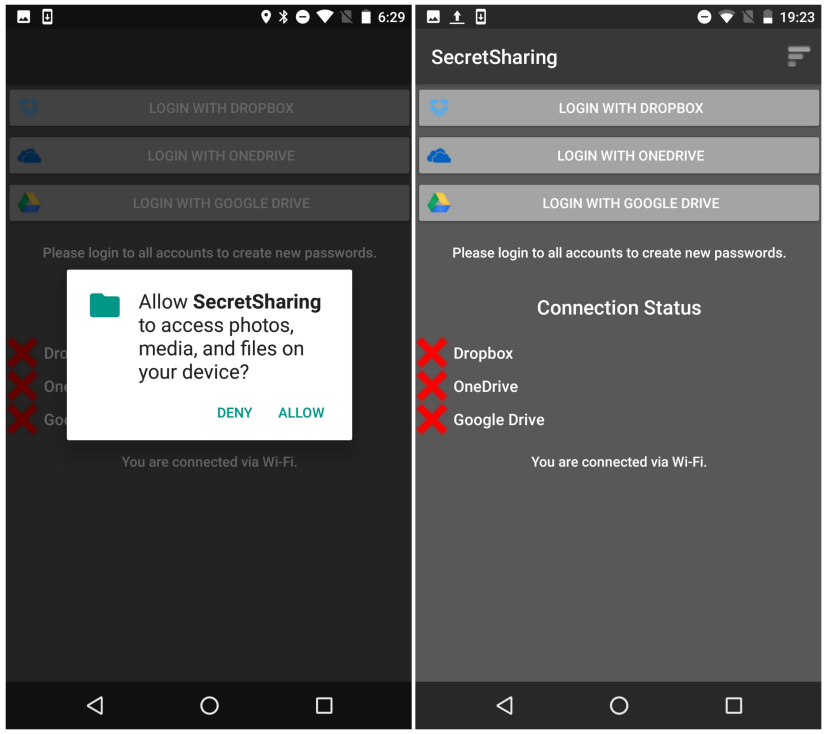

Figure 3.6: Screenshot of the application; left: verifying storage permissions, right: connection status page with no clouds connected and the device connected to the Internet via Wi-Fi.

The first time the application is opened, a pop-up will be presented to the user, asking for permission to access photos, media, and files on the device, as shown in Figure 3.6. This permission verification is needed to create the shares for the password, as the Dropbox shares are temporarily stored locally before being uploaded and deleted from the phone.

On the connection status page shown in Figure 3.6, the user will have an overview of whether the clouds are connected or disconnected, and also if the phone is connected to the Internet. If the user is not logged into to any of the clouds, all of the log-in buttons will be visible and all the clouds under "Connection Status" will be marked with a red cross, representing that it is not connected. A message below the clouds will indicate whether the phone is connected to Wi-Fi, a mobile network or disconnected from the Internet.

The user can log into the presented clouds with their personal user accounts. The red cross will then change to a green "OK" symbol, indicating that the application is connected to the particular cloud. Also, the log-in button will become invisible. The process of logging into the clouds is shown in Figure 3.7. When all clouds are connected, the message asking the user to log into all clouds to create a new password, shown in the first two screenshots of Figure 3.7, will become invisible.

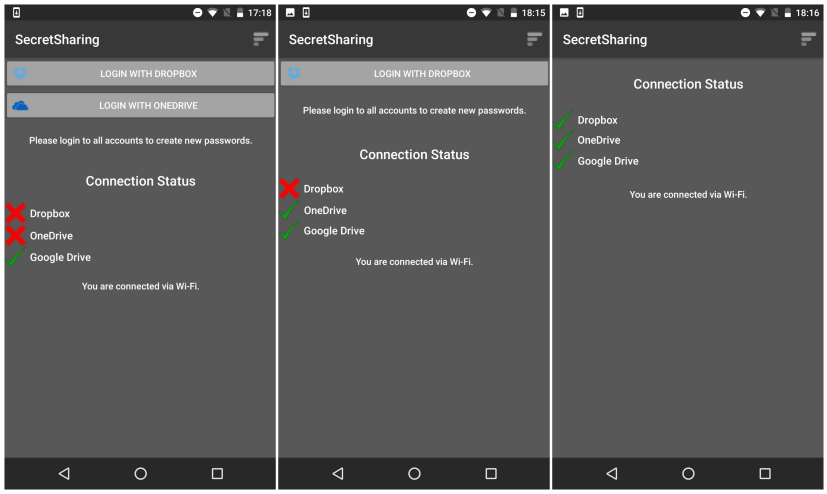

Figure 3.7: Screenshot of the application; left: Google Drive connected, middle: Google Drive and OneDrive connected, right: all clouds connected.

The Drop-down Menu

The access to the elements in the drop-down menu will change based on how many clouds that are connected, as shown in Figure 3.8. When all clouds are disconnected, as Figure 3.6 shows, the user will only have access to changing the password for the application. If one cloud is connected, the user will get access to changing the password for the application, as well as logging out from the cloud. If two clouds are connected, the user will have access to retrieving a password as well. The application uses a (2,3) threshold scheme, meaning that two clouds must be connected to retrieve a password. So by only having two clouds connected, the user should still have access to retrieving the passwords. The user will also have access to returning to the main page. If all clouds are connected, the user will have full access to the application and will be able to create a new password.

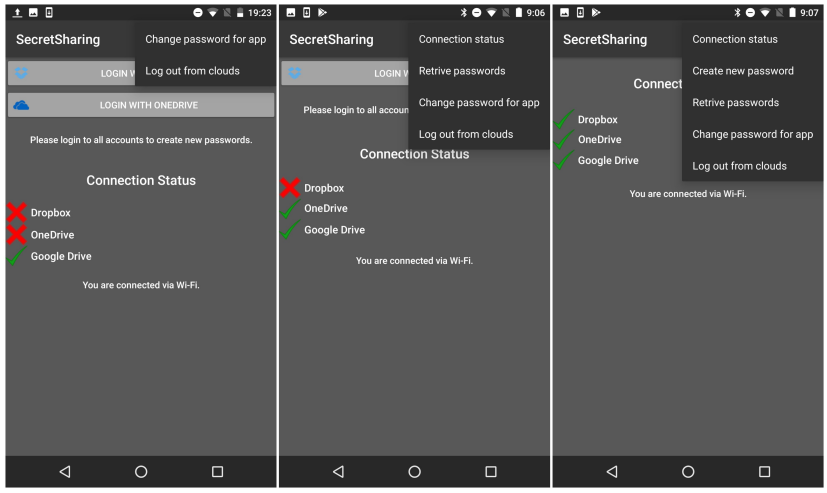

Figure 3.8: Screenshot of the application; left: drop-down menu when one cloud is connected, middle: drop-down menu when two clouds are connected, right: drop-down menu when all clouds are connected.

Disconnecting from the Cloud Services

The user can disconnect from the clouds by pressing the "Log out from clouds" button in the drop-down menu. The user will receive a pop-up with a warning message, asking if the user wants to remove the connection to the clouds, and by pressing "OK," the application will log out from all clouds connected as shown in Figure 3.9.

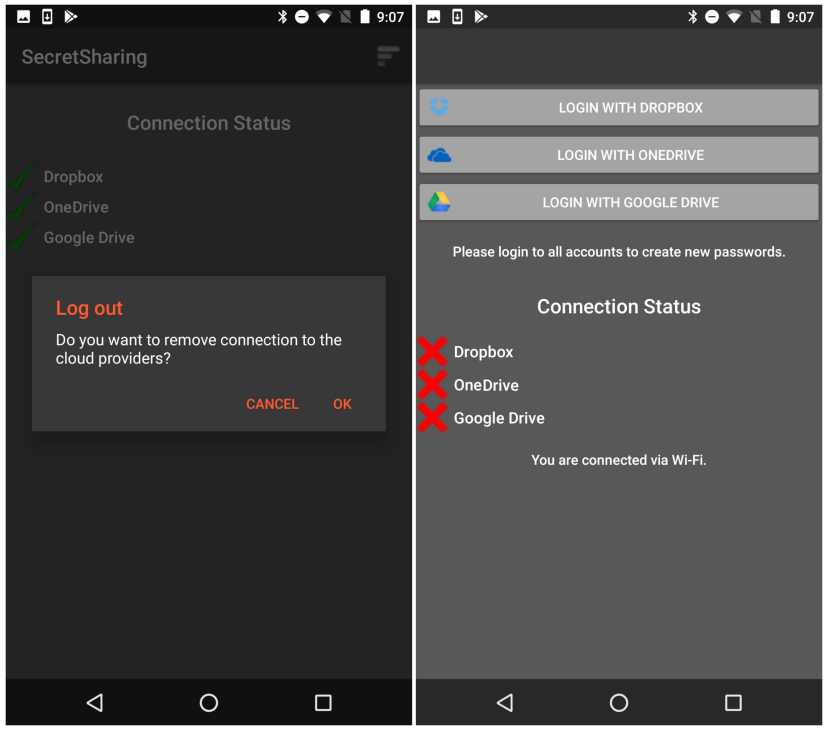

Figure 3.9: Screenshot of the application; left: warning message asking if the user wants to remove the connection to the clouds, right: connection status page with all clouds disconnected after being logged out.

3.2.3 Creating a New Password

When creating a new password, the user will have to press the "Create new password" button in the drop-down menu and by that be directed to the page shown in Figure 3.10. Here the UI, containing fields for the platform name and the password, as well as a save button, is displayed. The platform name is equal to PASSWORD-ID shown in Figure 3.2 in Section 3.1, and was renamed during development. The text in the password field is hidden by default. When the save button is pressed, the application will check whether both fields have content. If so, a pop-up is prompted, asking if the user wants to save the provided password for the provided platform name.

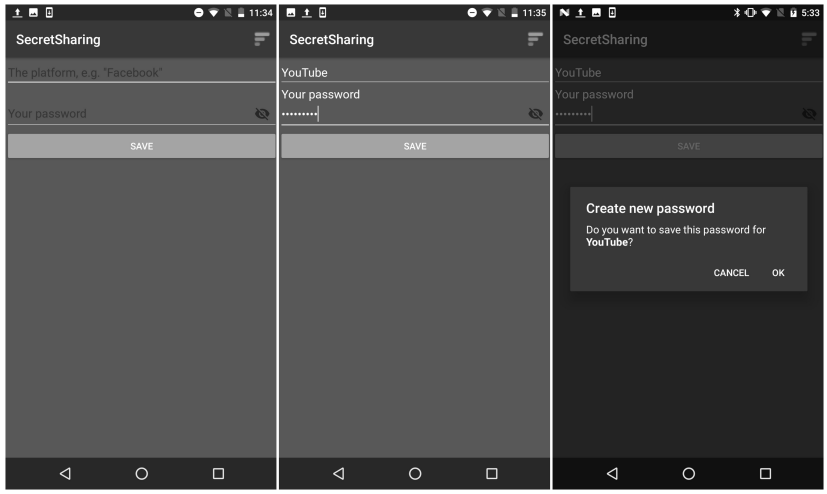

Figure 3.10: Screenshot of the application; left: create new password page, middle: filled in platform name and password, right: pop-up asking if user wants to save the password.

After the user has decided to save the password, the application will download the *backup files*. The backup files contain all existing platform names and are meant to act as a backup if the phone is lost or stolen, and is stored in two of the clouds to create redundancy. When creating a password, the files are used to prevent duplicates of a password, and the application will search through the files and compare the content with the platform name provided by the user. As shown in Figure 3.11, the user will get a message notifying that the backup file has been downloaded. However, if both of the downloads fails, or if one of the clouds is disconnected, the user will receive an error message asking the

user to try again. Only one of the backup files is needed, as the content should be identical, but since both clouds are required for uploading the shares, the connection must be up. If no backup files are found, the password is created, as long as it does not match any of the passwords saved in the local password list, shown in Figure 3.12.

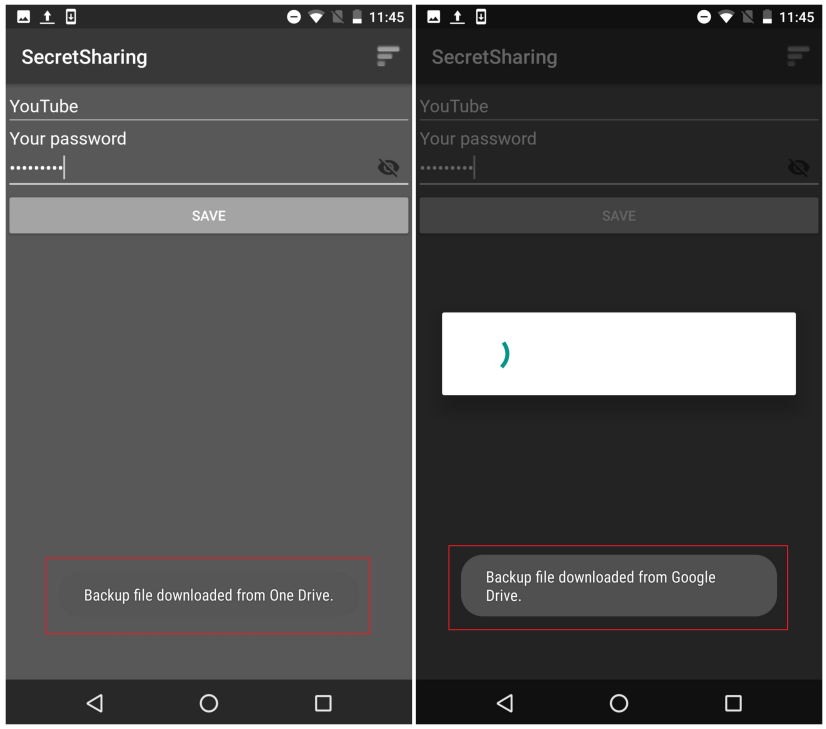

Figure 3.11: Screenshot of the application; left: backup file downloaded from OneDrive, right: backup file downloaded from Google Drive.

Each cloud will receive a share titled with the platform name, which will be used to recompute the password. Figure 3.12 shows the message the user will receive after the upload, notifying that the password is saved for the provided platform name. Also, to make sure that the backup files remain updated, the backup files will be updated on both of the drives.

Figure 3.13 shows the cloud contents after the upload is complete. The shares are stored on each of the clouds, and Google Drive and OneDrive contains the updated backup file titled "BACKUP." In this case, the content will be "Facebook" and "YouTube," in accordance with the shares.

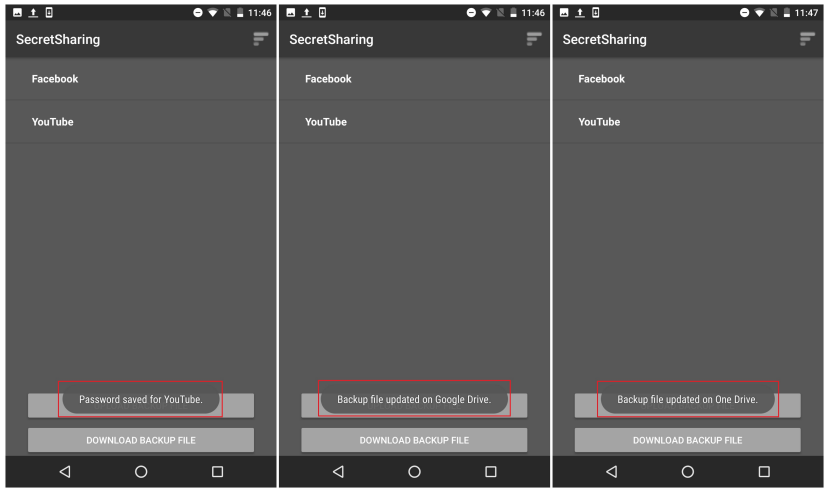

Figure 3.12: Screenshot of the application; left: retrieve passwords page with password saved, middle: backup file updated on Google Drive, right: backup file updated on OneDrive.

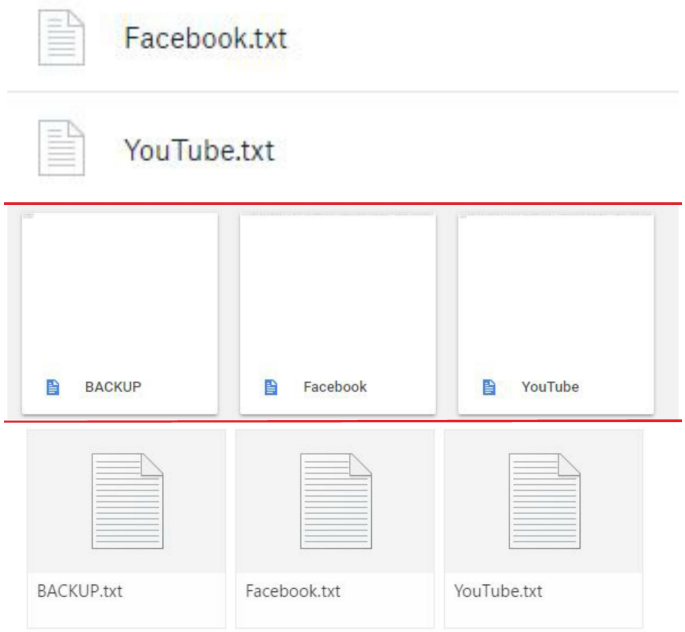

Figure 3.13: Screenshot of cloud content after password creation (the cloud interfaces are separated by red lines); upper: Dropbox content, middle: Google Drive content, lower: OneDrive content.

Measures for Preventing Overwriting
To prevent the shares and backup files from being overwritten, two main restrictions are set for the users.

1. The users can not create a password with a platform name that already exists (case insensitive).
2. The users can not create a password with the platform name "backup" in any form (case insensitive).

Figure 3.14 shows the error messages the user will receive when attempting to break these restrictions.

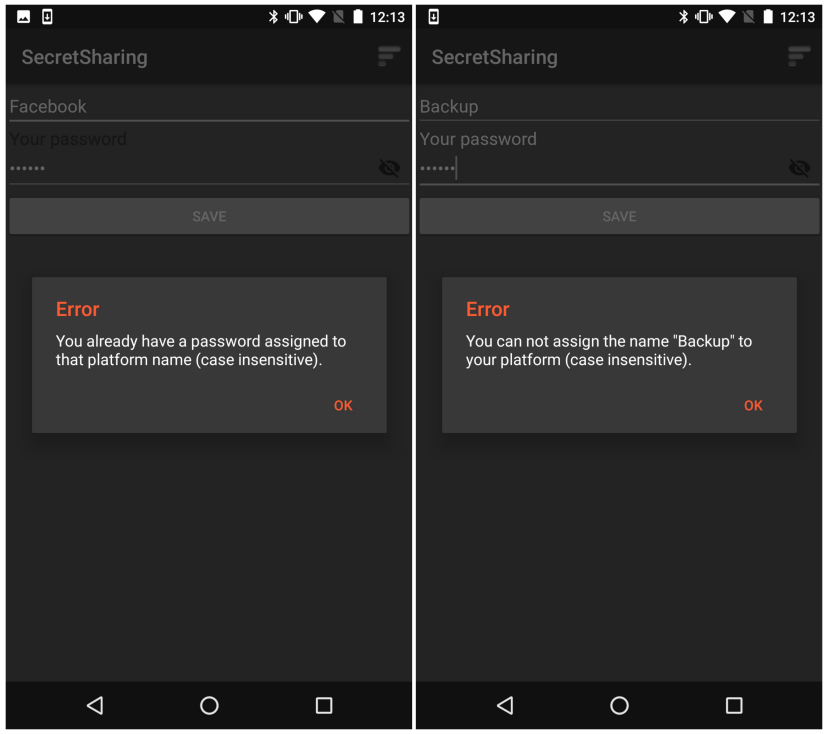

Figure 3.14: Screenshot of the application; left: error when attempting to create a password with a platform name already in the password list or in the backup files, right: error when attempting to create a password with the platform name "backup" in any form.

3.2.4 Retrieving a Password

Retrieving a password is relatively straight forward. By choosing "Retrieve passwords" in the drop-down menu, the application will display a page containing a list of the created passwords. The list also includes the platform names from the backup file, as long as these has been downloaded. The page is shown in Figure 3.15. The user clicks on the platform name that corresponds to the desired password, and by pressing "OK" on the pop-up, the shares will be downloaded and used to reconstruct the password, before displaying it to the user.

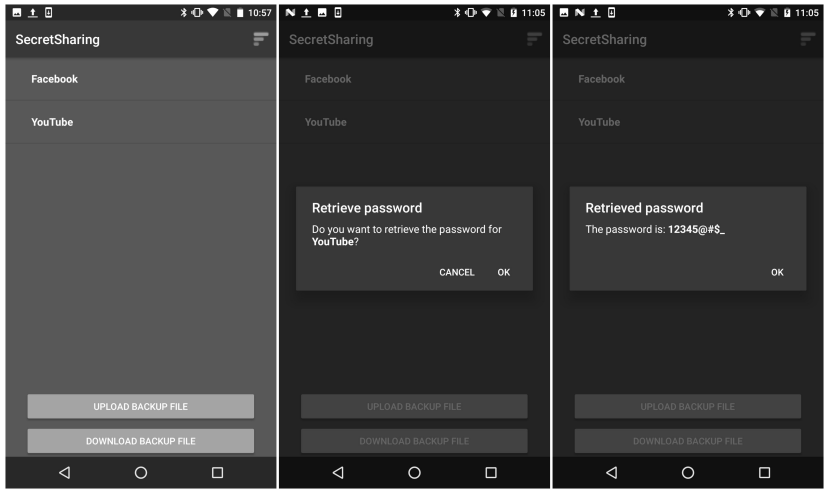

Figure 3.15: Screenshot of the application; left: retrieve passwords page, middle: pop-up asking if the user wants to retrieve the chosen password, left: the password retrieved.

3.2.5 Deleting a Password

When deleting a password, the user will have to click on the chosen platform name and hold for a short while, before a pop-up with a warning message is displayed, as shown in Figure 3.16.

After the user presses "OK," the application will delete all the shares corresponding to that password in the clouds and remove the platform name from the list. As shown in Figure 3.17, the user will be notified with individual messages for each of the clouds, indicating whether or not the share has been removed. Also, a pop-up informs the user that the password is deleted and that the user should remember to empty the trash cans in the clouds as well, in case the shares has not been removed correctly. After the password is deleted, the backup files will be updated, as shown in Figure 3.18.

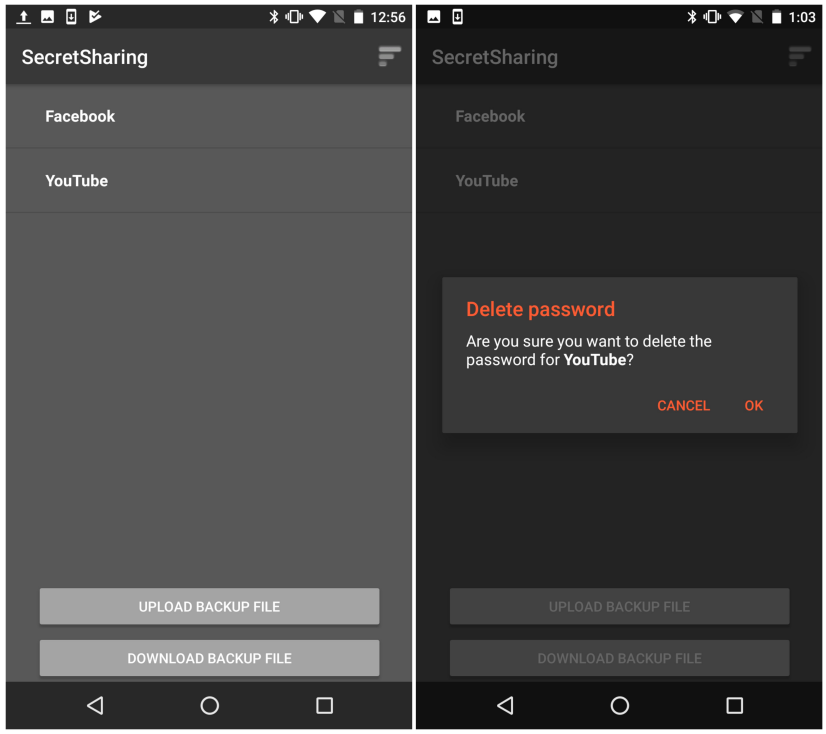

Figure 3.16: Screenshot of the application; left: retrieve passwords page, right: warning message for password deletion.

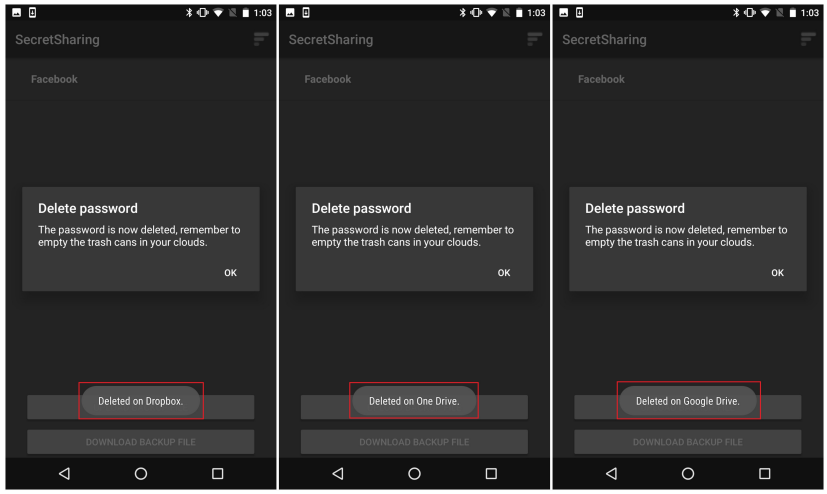

Figure 3.17: Screenshot of the application; left: share deleted on Dropbox, middle: share deleted on OneDrive, left: share deleted on Google Drive.

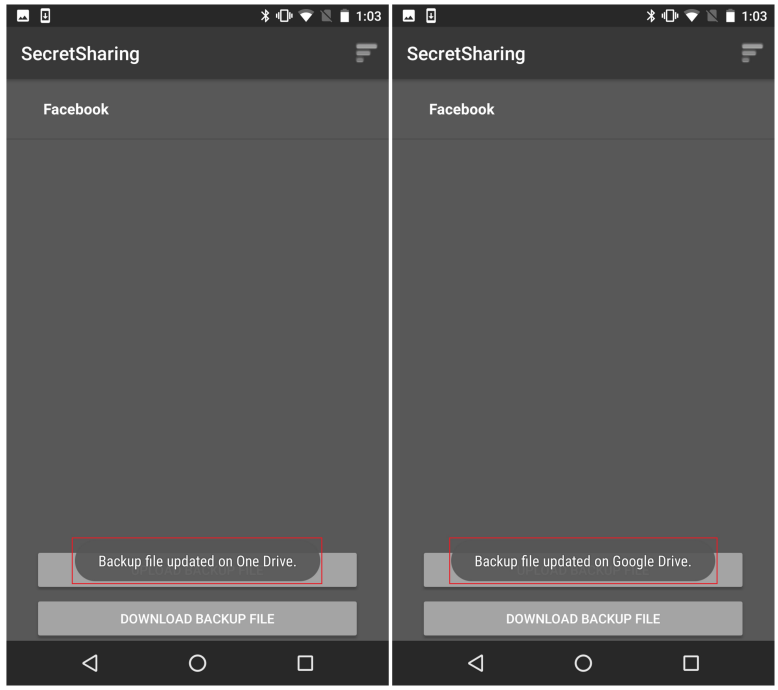

Figure 3.18: Screenshot of the application; left: backup file updated on OneDrive, right: backup file updated on Google Drive.

If the user goes on to delete the last password in the list, the application will remove the backup files as well. A message will be displayed, as shown in Figure 3.19, informing the user that there are no passwords in the list, and since there is no need for the backup files anymore, these are going to get deleted. When the files have been removed, the user will receive messages indicating if the files were deleted correctly.

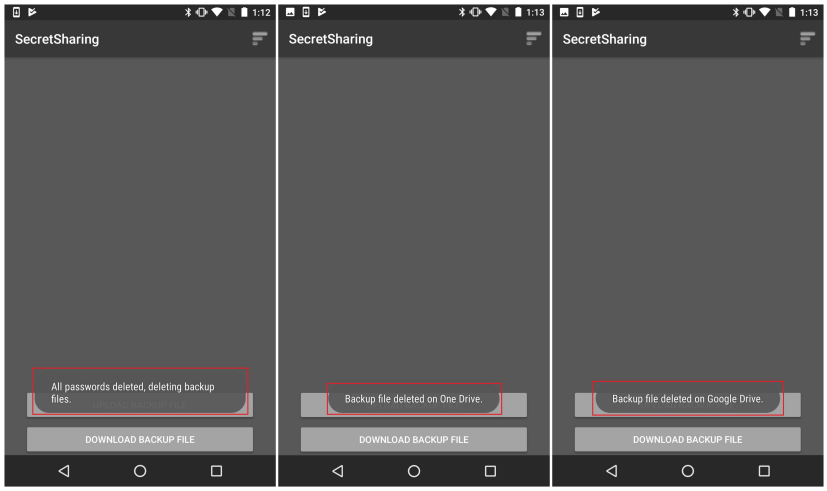

Figure 3.19: Screenshot of the application; left: deleting backup files when all passwords are deleted, middle: backup file deleted on OneDrive, right: backup file deleted on Google Drive.

3.2.6 Advanced Functionality

Advanced functionality is here defined as the features which are not part of the basic process of creating, retrieving and deleting passwords. This section includes the creation of a new application password, how the application reacts when losing access to the Internet, uploading and downloading the backup files manually, retrieving a password when one cloud is disconnected, and the notifications the user will receive when shares are deleted outside the application.

The functionality for retrieving a password when one cloud is disconnected is part of the core concept of the application, but it is not something that the user will experience during normal use. Therefore, it is placed under the advanced functionality section.

New Application Password

To create a new application password, which is the password created in Figure 3.4, the user has to press the "Change password for app" button in the drop-down menu. As shown in Figure 3.20, the user will receive a pop-up with a warning message about creating a new password for the application. By pressing the "OK" button, the user is directed to the page for creating a new password. The process of creating a new password and logging in is described in Section 3.2.1.

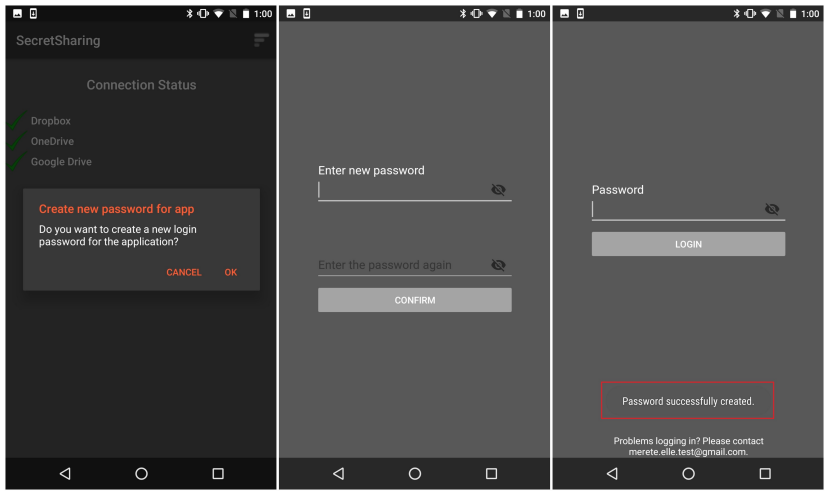

Figure 3.20: Screenshot of the application; left: warning message for creating new application password, middle: creating a new application password, right: login page - password successfully created.

No Access to the Internet

If the phone loses connection to the Internet, the user will not be able to use the application. Figure 3.21 shows the error messages displayed to the user when trying to e.g. create a new password or retrieve a password. Also, if the user opens the application without being connected to the Internet, all clouds will have a "NOT OK" status, and the user will not have access to the drop-down menu. The message below the clouds will inform the user that the phone is disconnected from the Internet and that it has to be connected to use the application. If the user tries to log out of the clouds while being disconnected from the Internet, this is the page that will get loaded. Then, when the phone reconnects to the Internet, all clouds will be logged out as shown in Figure 3.6.

The reason for implementing this is to reduce the probability of errors and ensure that all shares and backup files are delivered correctly, without potentially creating queues for when the phone is reconnected to the Internet.

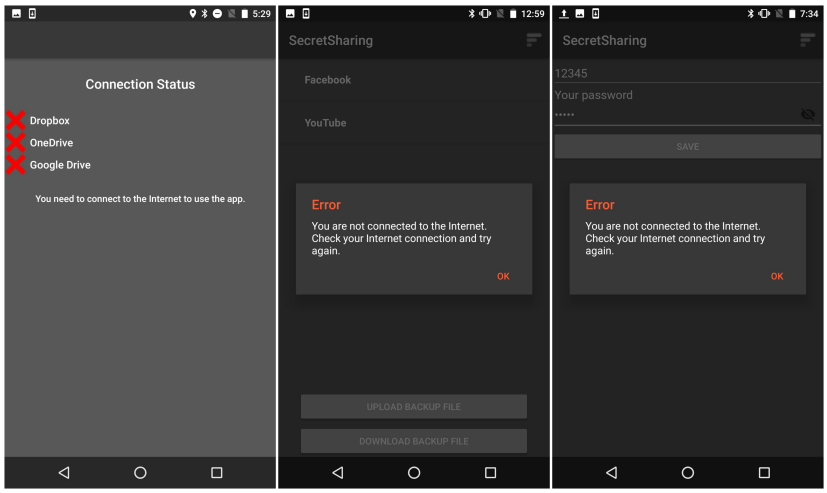

Figure 3.21: Screenshot of the application; left: connection status page with no Internet connection, middle: create new password page with no Internet connection, right: retrieve password page with no Internet connection.

Uploading and Downloading the Backup Files Manually
The user also has the possibility to upload and download the backup files manually. This feature is implemented so that the user do not have to create a new password to download the backup files, in case, for example, one of the clouds is disconnected. The pop-ups the user will receive are referring to a single backup file, as the content will, in most cases, be identical, and from the users perspective, it is seen as a single entity.

By clicking on the "Upload backup file" button on the retrieve passwords page, the user will receive the pop-up shown in Figure 3.22. If a backup file has already been created, the user will also receive a pop-up with a warning message, informing the user about the possibility of losing passwords. After pressing the "OK" button, the user will receive the same messages as in Figure 3.18, indicating that the backup files have been updated on OneDrive and Google Drive.

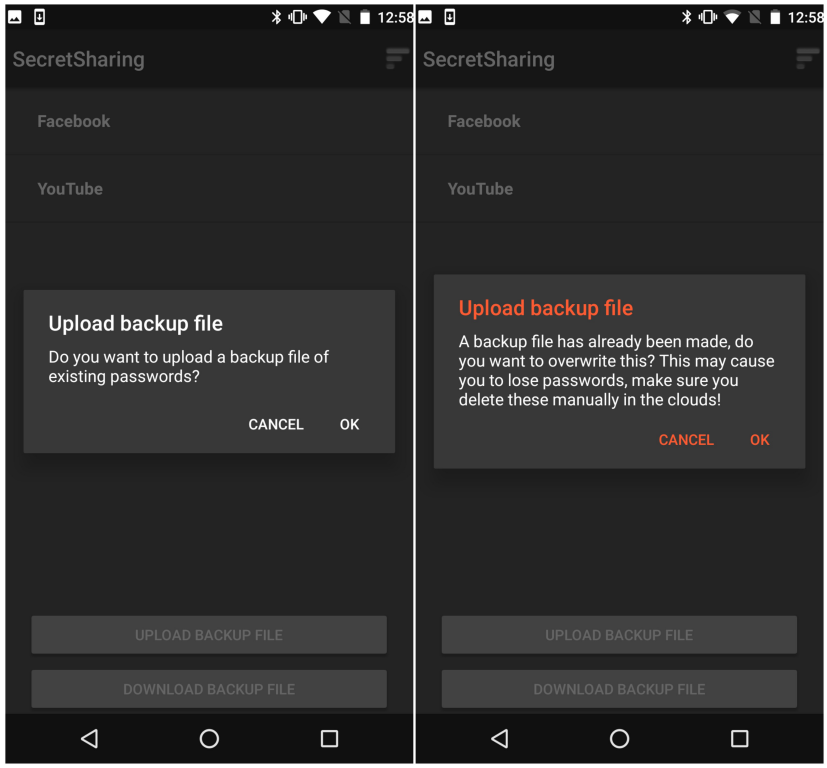

Figure 3.22: Screenshot of the application; left: pop-up asking if the user wants to upload a backup file, right: warning message informing about the possibility of losing passwords.

To download a backup file manually, the user will have to click the "Download backup file" button on the retrieve passwords page. Similar to uploading the backup file, the user will receive a pop-up, asking to download the backup file. By pressing the "OK" button, the backup files will be downloaded from OneDrive and Google Drive, as long as the files exist. Figure 3.23 shows the messages the user will receive, which includes instructions to reload the page to update the list of platform names.

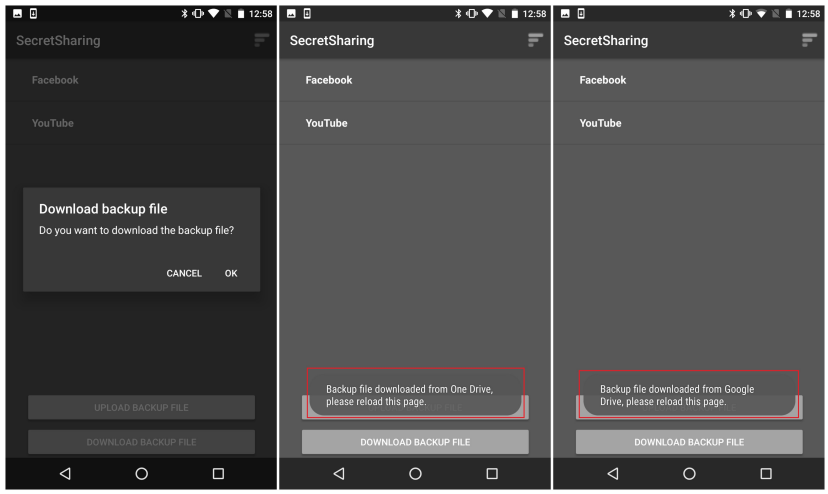

Figure 3.23: Screenshot of the application; left: pop-up asking if the user wants to download the backup file, middle: backup file downloaded from OneDrive, left: backup file downloaded from Google Drive.

One Cloud Disconnected

Since the application uses a (2,3) threshold scheme, described further in Chapter 4, Section 4.5, the user should still be able to retrieve passwords even though one of the three clouds is disconnected. To delete or create new passwords, the user has to be logged into all clouds, as this affects all clouds.

Figure 3.24 shows the procedure of retrieving a password when one cloud is disconnected, and it is identical to the process described in Section 3.2.4.

Furthermore, as an example, Figure 3.25 shows the message the user will receive when uploading the backup files with one cloud disconnected. Here, the warning message about overwriting the previously saved backup file, shown in Figure 3.22, is not included. When downloading the backup

files, the application will display a similar message informing the user that OneDrive is not connected.

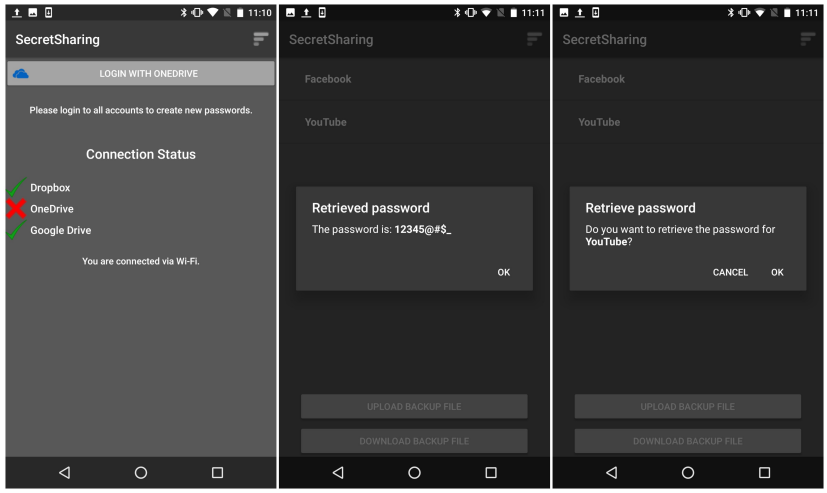

Figure 3.24: Screenshot of the application; left: OneDrive disconnected, middle: pop-up asking to retrieve password, right: password retrieved.

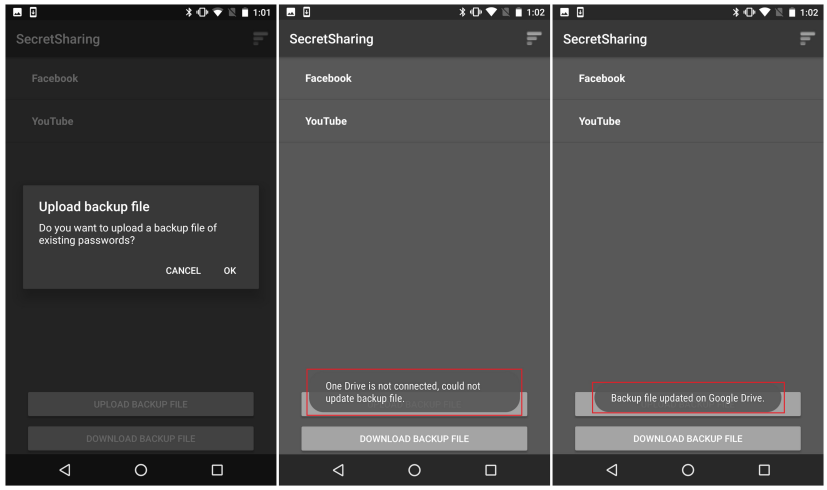

Figure 3.25: Screenshot of the application; left: pop-up asking if the user wants to upload a backup file, middle: could not update backup file on OneDrive, right: backup file updated on Google Drive.

Shares Deleted on the Clouds

Since clouds are used for storing the shares, there is a certain chance that the shares can be deleted outside the application. If the shares are removed from the clouds, an error message will be displayed, as shown in Figure 3.26. This message informs the user that there are not enough shares available to reconstruct the password.

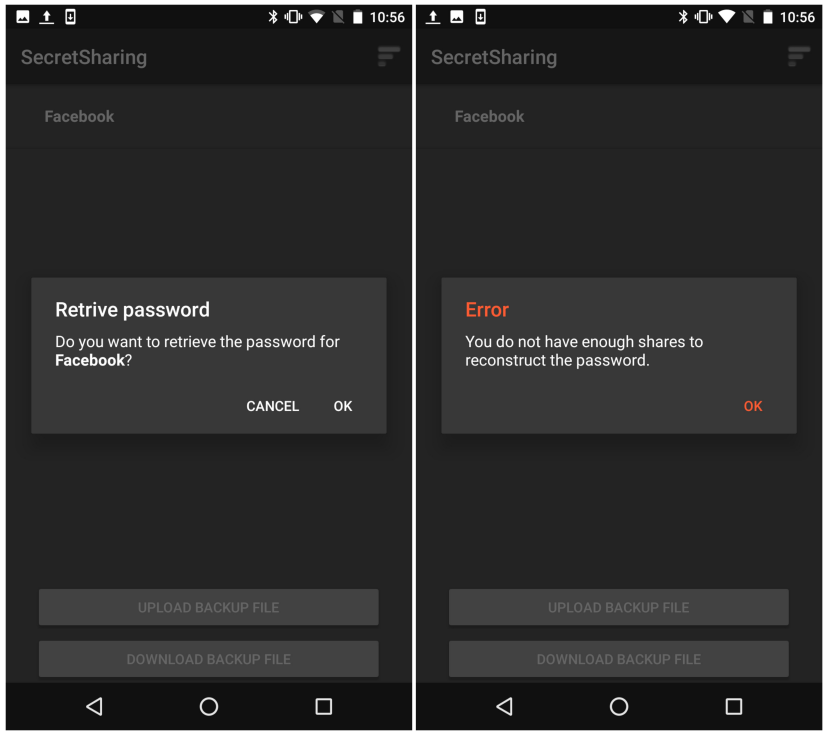

Figure 3.26: Screenshot of the application; left: pop-up asking if the user wants to retrieve the password, right: error while retrieving the password - did not find enough shares.

If the user tries to delete a password that has no shares, a message will notify the user that no shares were found and that the password has been deleted, as shown in 3.27. As described in Figure 3.18, the application will give the user feedback on whether or not the share was deleted. If no share were found, the user would receive a notification informing that an error occurred while deleting the share, and ask the user to remove the share manually if it exists.

For this case, the deleted password is the last password in the list, so the backup files will also be deleted as shown in Figure 3.19.

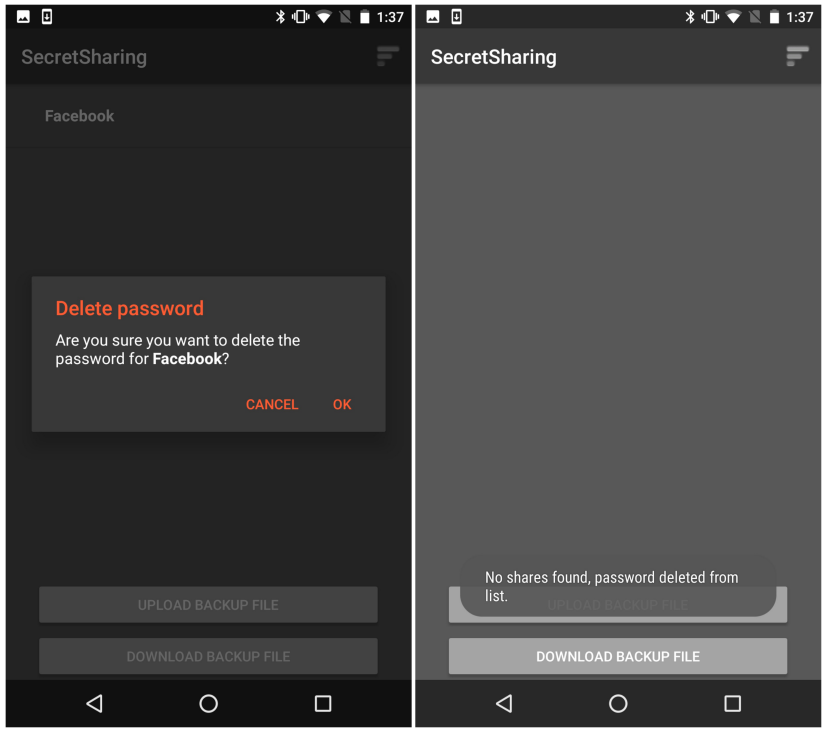

Figure 3.27: Screenshot of the application; left: warning message asking if the user wants to delete the password, right: no shares found - deleting password from list.

Chapter 4
The Functionality of the *SecretSharing* Application

This chapter describes the functionality of the application. First, a short introduction is given, before the Android manifest for the application is presented. Then, the functionality for connecting to the cloud APIs is described, before the activities for the application is explained. Lastly, the implementation of secret sharing is presented. The activities are primarily described through activity diagrams, while the rest is described using mainly code.

4.1 Introduction

The password storage application is named "Secret Sharing" and uses Shamir's secret sharing algorithm to divide the password into three shares, before storing these on cloud storage services.

As presented in the introduction, the official IDE for Android, Android Studio, is used for developing the application, with Java as the programming language [Goo17b]. The code is not included in the appendices because of the number of pages required but uploaded to a repository on GitHub. It can be found by using the following URL: `https://github.com/meretele/SecretSharing`. However, pieces of the code are included in the following sections for illustrating purposes.

4.2 The Android Manifest

As described in Chapter 2, Section 2.3, the Android Manifest includes important information about the application, including the minimum API level required, identified user permissions, and declaration of all components.

Listing 4.1: The permissions in the *AndroixManifest.xml* file.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission
    android:name="android.permission.MANAGE_DOCUMENTS"/>
////
 <meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="[YOUR API KEY]" />
////
<activity
    android:name="com.android.merete.secretsharing.SplashScreen"
    android:label="@string/app_name"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.Black.NoTitleBar" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
////
<activity
    android:name="com.dropbox.core.android.AuthActivity"
    android:configChanges="orientation|keyboard"
    android:launchMode="singleTask" >
    <intent-filter>
        <data android:scheme="db-[YOUR APP KEY]" />
        <action android:name="android.intent.action.VIEW" />
        <category
            android:name="android.intent.category.BROWSABLE" />
        <category
            android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Listing 4.1 shows some parts of the manifest file for the application. First, the permissions are listed. These includes access to the Internet, the network state, reading and writing to the external storage, and managing

access to documents. A list of all permissions available can be found on the Android Developers reference pages [Goo17l]. Then, additional metadata added are listed, which here includes the API key for the Google API. Next, an activity is listed and this is the entry point for the application, as indicated by *<action android:name="android.intent.action.MAIN"/>*, and the last activity listed is the authentication activity for Dropbox, containing the application key.

4.3 Connecting to the APIs

As described in Chapter 2, Section 2.6, a cloud storage API connects a locally-based application to a cloud-based storage system. For the thesis, Dropbox, Google Drive, and Microsoft OneDrive are the cloud storages used. The following subsections will explain the code for connecting to the clouds.

4.3.1 Dropbox

The code used for connecting to the Dropbox API is provided by the Dropbox Core Software Development Kit (SDK) for Java 6+ [Dro17b].

Listing 4.2 displays the code for what happens when a user presses the "Login with Dropbox" button on the connection status page described in Chapter 3, Section 3.2.2. The *startOAuth2Authentication* method will start the OAuth 2.0 authentication activity for Dropbox, using the application key specific for this application. Listing 4.3 shows the code for this method, which is provided by Dropbox. This will prompt the user to log into Dropbox. OAuth 2.0 is described in Chapter 2, Section 2.6.

Listing 4.2: Logging into Dropbox, code from *UserActivity.java* file.

```
// Dropbox login.
    DBlogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Auth.startOAuth2Authentication(UserActivity.this,
                getString(R.string.app_key));
        }
    });
```

Listing 4.3: Dropbox authentication, code from *Auth.java* file provided by Dropbox.

```
public static void startOAuth2Authentication(Context context,
    String appKey) {
    if (!AuthActivity.checkAppBeforeAuth(context, appKey, true
        /*alertUser*/)) {
        return;
    }

    // Start Dropbox auth activity.
    String apiType = "1";
    String webHost = "www.dropbox.com";
    Intent intent = AuthActivity.makeIntent(context, appKey,
        webHost, apiType);
    if (!(context instanceof Activity)) {
        // If starting the intent outside of an Activity, must
            include
        // this. See startActivity(). Otherwise, we prefer to
            stay in
        // the same task.
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    }
    context.startActivity(intent);
}
```

4.3.2 Google Drive

The code used for connecting to the Google Drive API for Android is provided by the sample code from Google [Goo17e].

Listing 4.4 displays the code for what happens when a user presses the "Login with Google Drive" button on the connection status page described in Chapter 3, Section 3.2.2. A Google API client, *GoogleApiClient*, is created, which is the main entry point for interacting with the API. Then, by *mGoogleApiClient.connect();*, the authorization begins. If a user has not previously authorized the application, the user will be prompted to allow it to access the user's files in Google Drive [Goo17e].

Listing 4.4: Logging into Google Drive, code from *UserActivity.java* file.

```
// Google Drive login.
    GDlogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mGoogleApiClient = new
                GoogleApiClient.Builder(UserActivity.this)
                    .addApi(Drive.API)
                    .addScope(Drive.SCOPE_FILE)
                    .addConnectionCallbacks(UserActivity.this)
                    .addOnConnectionFailedListener(UserActivity.this)
                    .build();
            mGoogleApiClient.connect();
            }
        });
```

Listing 4.5 shows the *onConnected* method called asynchronously when the connect request has successfully completed. *SharedPreferences* saves data in a persistent key-value pair, and is used in the application to keep track of which clouds that have been connected. The application checks whether the application has been logged in before and if not, the string "Logged_in" is added to the shared preferences and the *onResume;* callback is called.

Listing 4.5: Connected to Google Drive, code from *UserActivity.java* file.

```
// If Google Drive is connected.
    @Override
    public void onConnected(Bundle connectionHint) {
        sharedpreferences = getSharedPreferences(mypreference,
            Context.MODE_PRIVATE);
        if (!sharedpreferences.contains(LoggedIn)) {
            String n = "Logged_in";
            SharedPreferences.Editor editor =
                sharedpreferences.edit();
            editor.putString(LoggedIn, n);
            editor.apply();
            onResume();
        }
    }
```

4.3.3 Microsoft OneDrive

The code used for connecting to the OneDrive API for Android is provided by the sample code from the OneDrive SDK [Mic17].

Listing 4.6 displays the code for what happens when a user presses the "Login with OneDrive" button on the connection status page described in Chapter 3, Section 3.2.2. The *createOneDriveClient* method is called to create the client, and uses a callback for the result, on whether it is a failure or success.

Listing 4.6: Logging into OneDrive, code from *UserActivity.java* file.

```
// OneDrive login.
    ODlogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            final ICallback<Void> serviceCreated = new
                DefaultCallback<>(UserActivity.this);
            createOneDriveClient(UserActivity.this,
                serviceCreated);
            }
        });
```

Listing 4.7 shows the creation of the client, and, as explained in Section 4.3.2 about connecting to the Google Drive API, a string is added to a key-value pair in the shared preferences if the creation is successful. The configuration for the client is not listed, but it includes the client ID specific for the application, as well as the scopes for the application.

Listing 4.7: Creating a OneDrive client, code from *UserActivity.java* file.

```
// Create One Drive client.
   synchronized void createOneDriveClient(final Activity
       activity, final ICallback<Void> serviceCreated) {
      final DefaultCallback<IOneDriveClient> callback = new
          DefaultCallback<IOneDriveClient>(activity) {
         @Override
         public void success(final IOneDriveClient result) {
            mClient.set(result);
            sharedpreferences =
                getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);
            SharedPreferences.Editor editor =
                sharedpreferences.edit();
            editor.putString(LoginOD, "OK");
            editor.apply();

            serviceCreated.success(null);
            onResume();
         }

         @Override
         public void failure(final
             com.onedrive.sdk.core.ClientException error) {
            serviceCreated.failure(error);

            sharedpreferences =
                getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);
            sharedpreferences.edit().remove(LoginOD).apply();
         }
      };
      new OneDriveClient
             .Builder()
             .fromConfig(createConfig())
             .loginAndBuildClient(activity, callback);
   }
```

4.4 Activities

An activity serves as the entry point for an applications interaction with the user, representing a single screen with a UI as described in Chapter 2, Section 2.2 [Goo17c]. The application contains six activities visible to the user, as well as one abstract class that is used as a base for activities that requires authentication tokens for Dropbox. All activities, excluding the splash screen, the login activity and the activity for creating an application password, use this base functionality.

The subsections of this section will present the activity diagrams for the activities developed for the application. Activity diagrams are behavior diagrams and give a graphical representation of the processes within the application, and is part of the Unified Modeling Language (UML). The *DropboxActivity* activity is not included, as this was provided by Dropbox [Dro17b]. For the creating and retrieving of passwords, sequence diagrams are given to provide an understanding of the communication between the entities. Sequence diagrams are also a part of UML. Figure 4.1 shows the relationship between the activities, and also includes *DropboxActivity*.

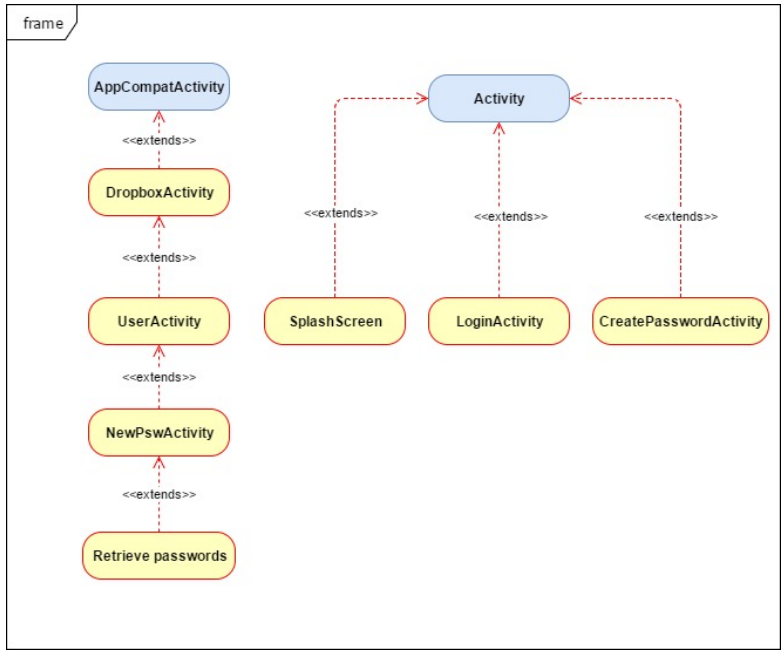

Figure 4.1: Relationship of the activities in the application.

Figure 4.2 presents the activities and menu options shown to the user, based on the number of clouds the user is logged into. All activities may terminate the application. This figure does not include all details, including e.g. the number of login attempts, which is described further in their respective subsections.

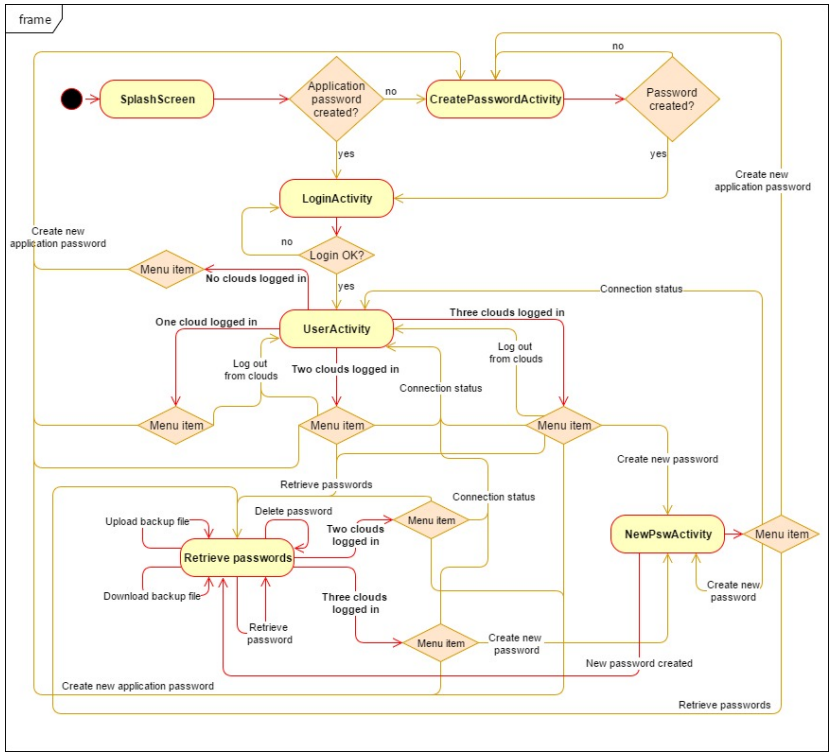

Figure 4.2: An overall representation of the activities and menu options shown to the user.

4.4.1 Splash Screen

The splash screen, as shown in Chapter 3, Section 3.2.1, is commonly known as a "welcome screen". Figure 4.3 shows the workflow of the activity named *SplashScreen*, where a timer will start once the application is opened. If an application password already exists, an intent will be sent to *LoginActivity* to start the login. Otherwise, the intent will be sent to *CreatePasswordActivity* to create an application password.

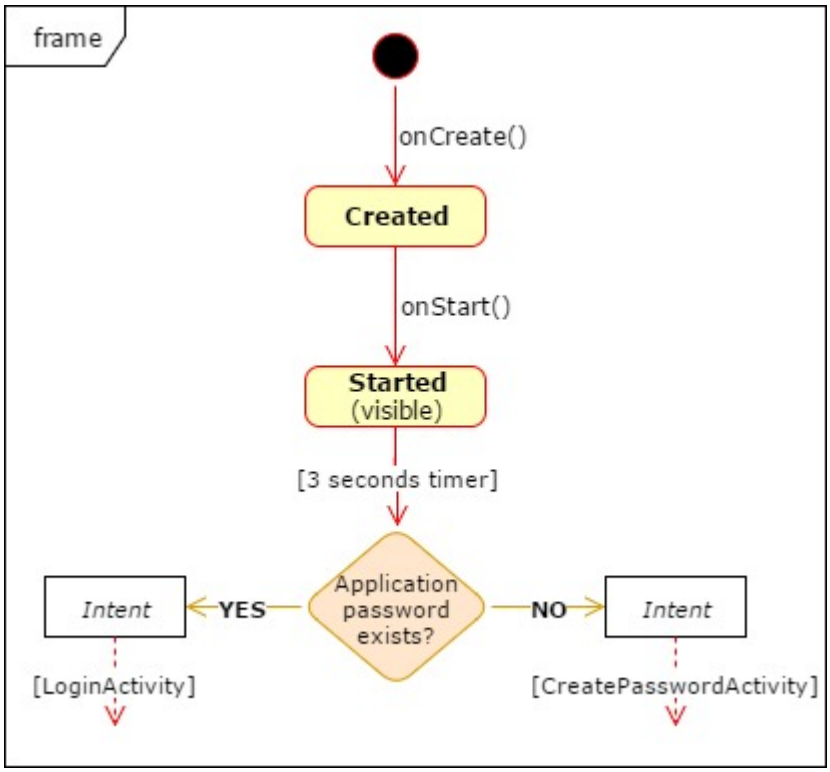

Figure 4.3: Activity diagram for the *SplashScreen.java* file.

4.4.2 Create Application Password

The *CreatePasswordActivity* is the activity for creating an application password, shown in Chapter 3, Section 3.2.1, and is shown the first time the user opens the application or if the item "Change password for app" in the drop-down menu is chosen.

Figure 4.4 shows the activity diagram for the activity, starting with an intent received from either *SplashScreen* or *UserActivity*. The variables

password1 and **password2** are the password fields shown to the user, where the first one is the upper field and the latter is the lower field. If the text in **password1** changes, the application will check whether the field contains characters. If so, the password strength is calculated and shown to the user. If the text changes in **password2** and the field contains characters, the **confirm** button is enabled. When this is clicked, the application will check whether both fields contain characters, and if they do, it will check if the text in the two fields is equal. If the fields are equal, the password will get saved as long as the hashing of the password does not fail, and an intent is sent to *LoginActivity* to start the login.

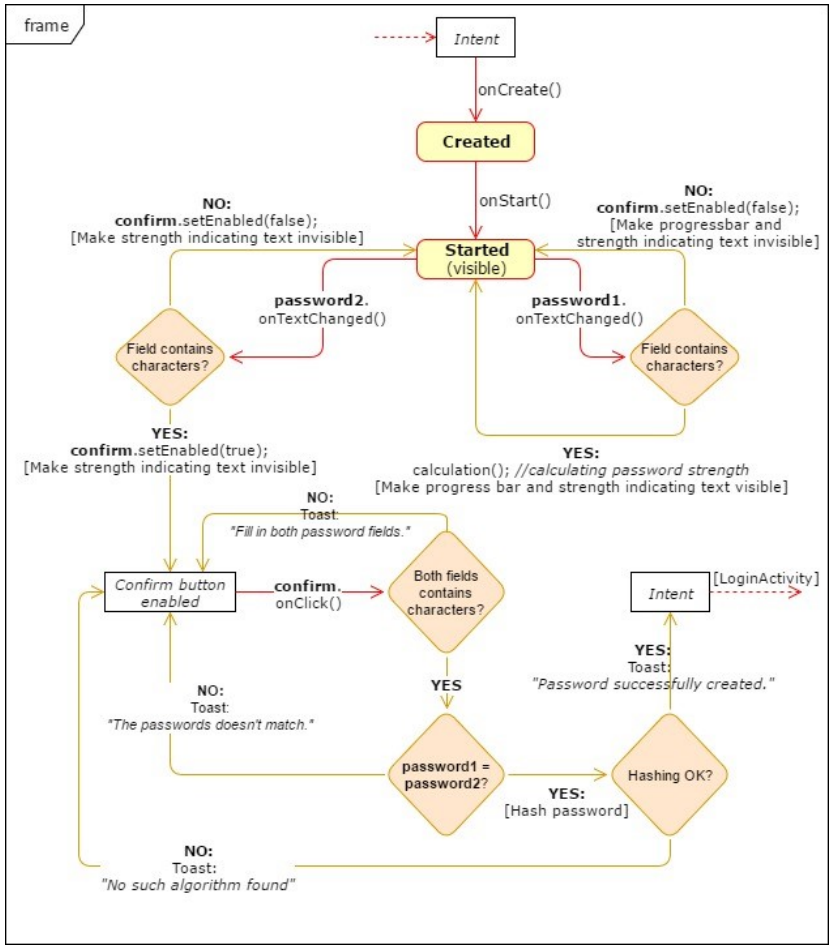

Figure 4.4: Activity diagram for the *CreatePasswordActivity.java* file.

4.4.3 Logging into the Application

The *LoginActivity* is the activity for logging into the application, and is shown in Chapter 3, Section 3.2.1. This page is shown if an application password exists when the application is opened, or right after an application password has been created.

Figure 4.5 shows the workflow of the activity, starting with an intent from either the *SplashScreen* or *CreatePasswordActivity*. When the text in the password field, **password**, changes, the **login** button will be enabled as long as the field contains characters.

If **login** button is pressed, a counter will add one to its value and check if the value is over or equal to five. If so, the maximum number of login attempts is reached, and the user will have to re-open the application to try again. If not, **password** is hashed and if this is successful, the password created in *CreatePasswordActivity* is compared to **password**. If these are equal, an intent will be sent to *UserActivity*.

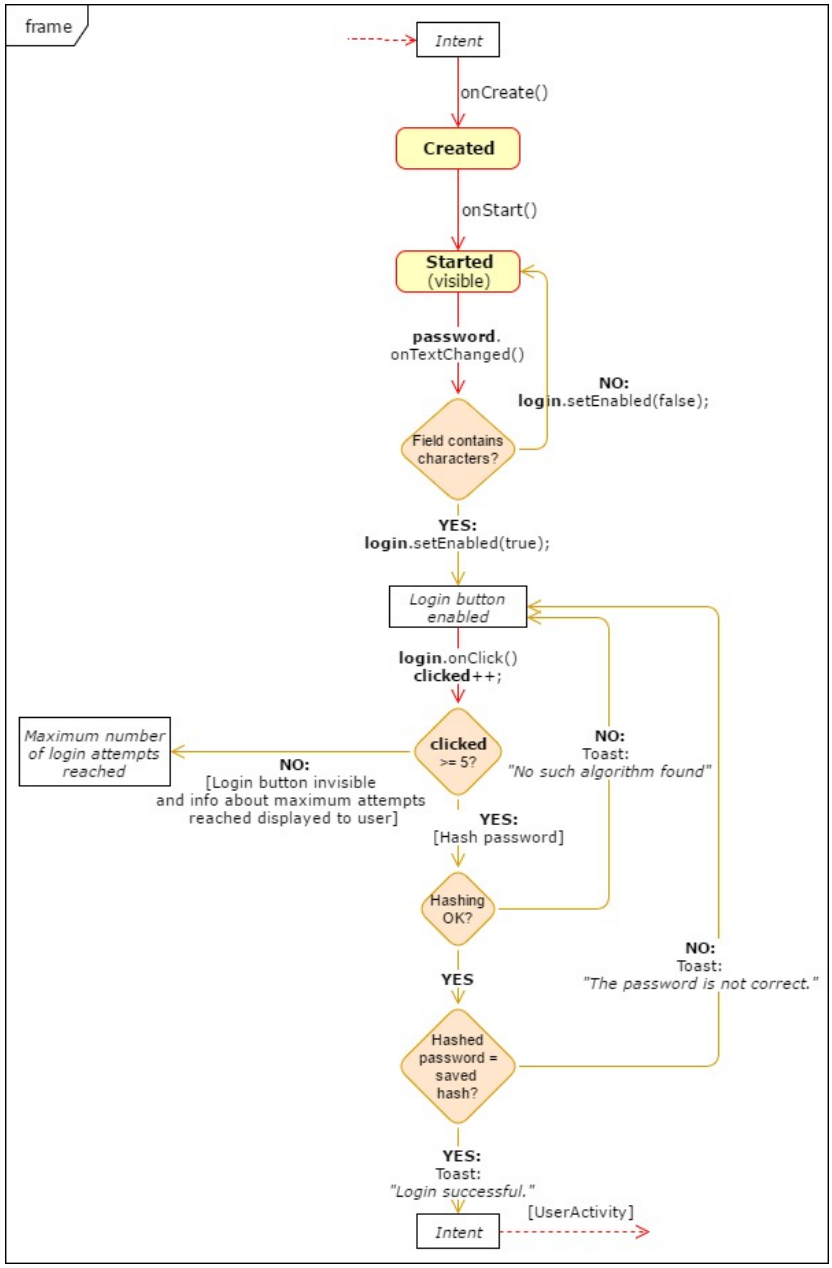

Figure 4.5: Activity diagram for *LoginActivity.java* file.

4.4.4 Connection Status

The *UserActivity* is the activity displaying the connection status of the application, and is shown in Chapter 3, Section 3.2.2. This page is shown when a user logs in, or if the item "Connection status" in the drop-down menu is chosen.

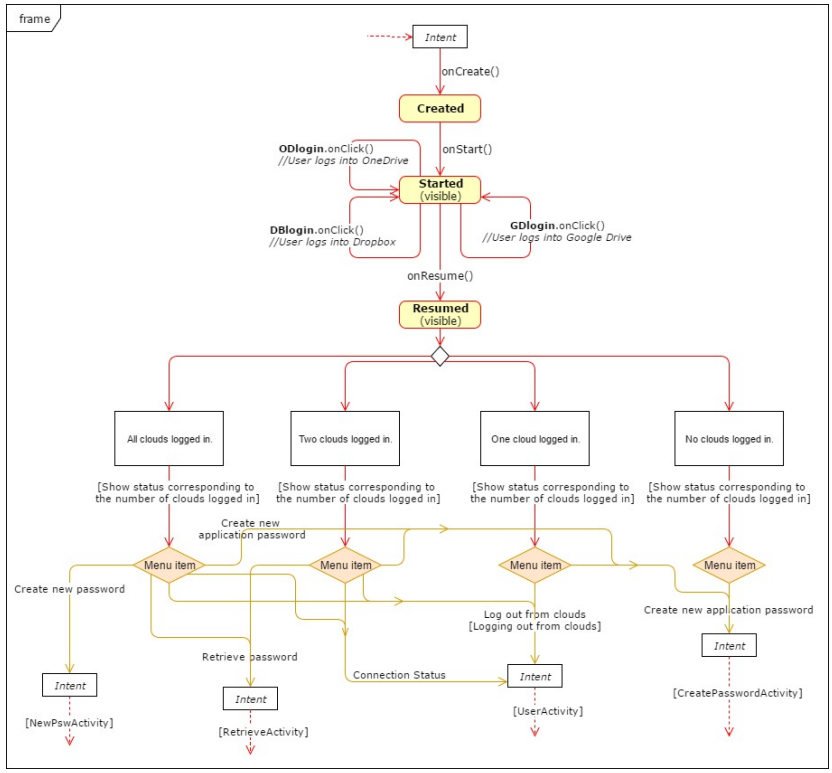

Figure 4.6: Activity diagram for *UserActivity.java* file.

Figure 4.6 shows the behavior diagram of the activity. The activity is started by an intent from one of the other activities, and the first time a user sees this site, all login buttons will be visible. By pressing one of the buttons, the user is prompted to log into the specific cloud. If the login is successful, the button becomes invisible, and the status for that cloud is "OK". The four, white boxes in the diagram symbolize the different scenarios in regards to how many clouds that are logged in. This is relevant since the number of connected clouds is the factor deciding how many menu items the user can see. The menu items for each of the scenarios is shown, where the scenario with all clouds being logged in is

the only one having full access. To avoid having a less detailed and larger diagram, the checking of network connectivity and the specific variables is not included.

4.4.5 Create New Password

The *NewPswActivity* is the activity for creating a new password to be saved to the application, and is shown in Chapter 3, Section 3.2.3. This page is shown if the item "Create new password" in the drop-down menu is chosen.

Figure 4.7 shows the workflow of the activity. As in Section 4.4.4 the checking of network connectivity and specific variables is not included to avoid having a less detailed and larger diagram. The activity is started by an intent and displays the page for creating a new password. When the button for saving the password is clicked, the application will check if both fields contain characters. If so, it will check whether the platform name is "backup" in any form. If it is not, the backup files, if they exist, will be downloaded from the clouds. Otherwise, if both fields do not contain characters or the platform name is "backup," error messages is shown to the user.

When the backup files are downloaded, the application will check if the platform name is equal to a platform name that already exists - including the ones in the backup files. If it already exists, an error message is shown to the user. If not, the application will create the shares and upload them to the clouds. Then, if the upload is successful, an intent is sent to *RetrieveActivity* containing the platform name.

Figure 4.8 shows the simplified sequence diagram for the activity. It does not include all error messages and details but is provided to give an understanding of the communication between the entities.

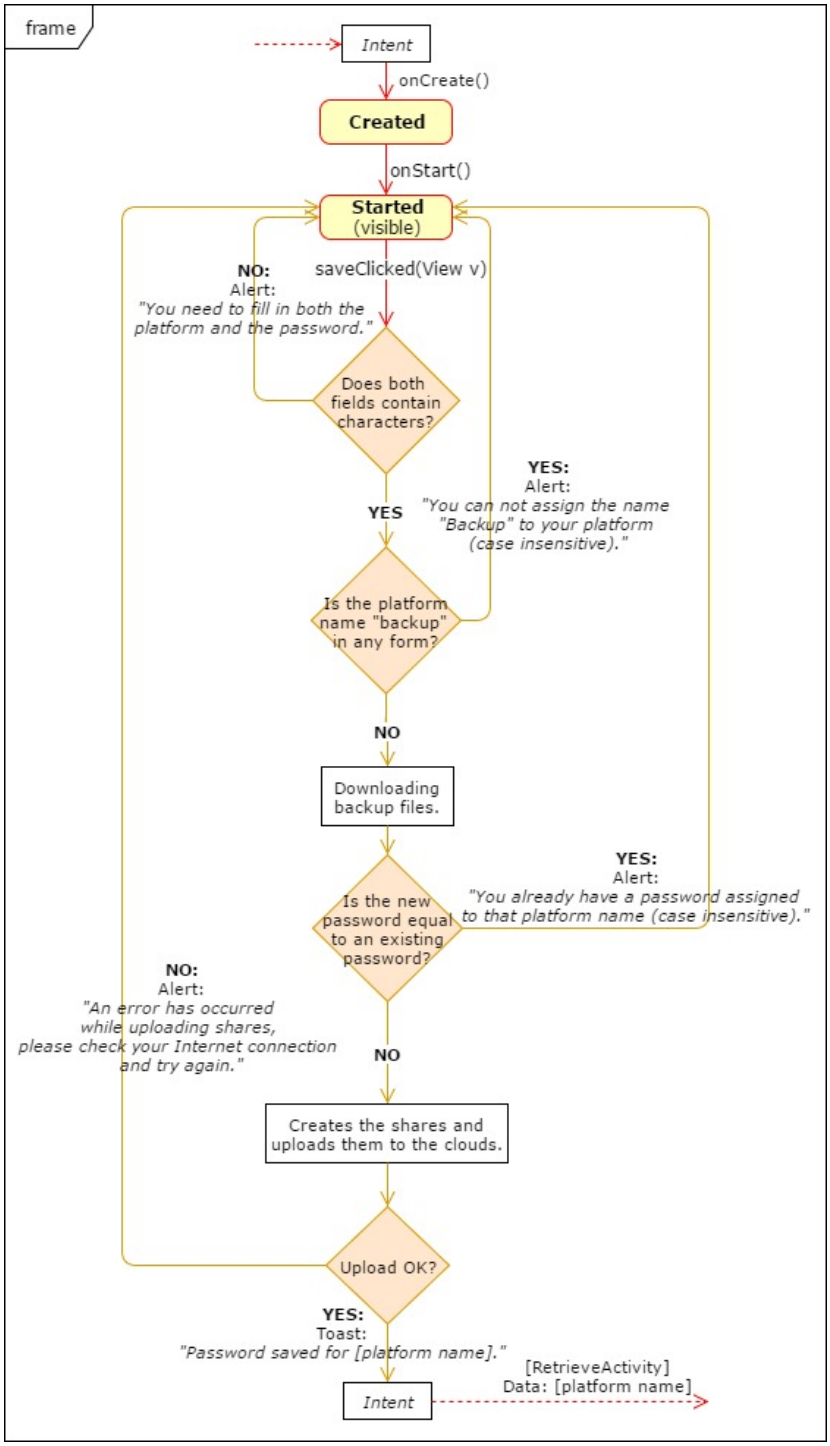

Figure 4.7: Activity diagram for *NewPswActivity.java* file.

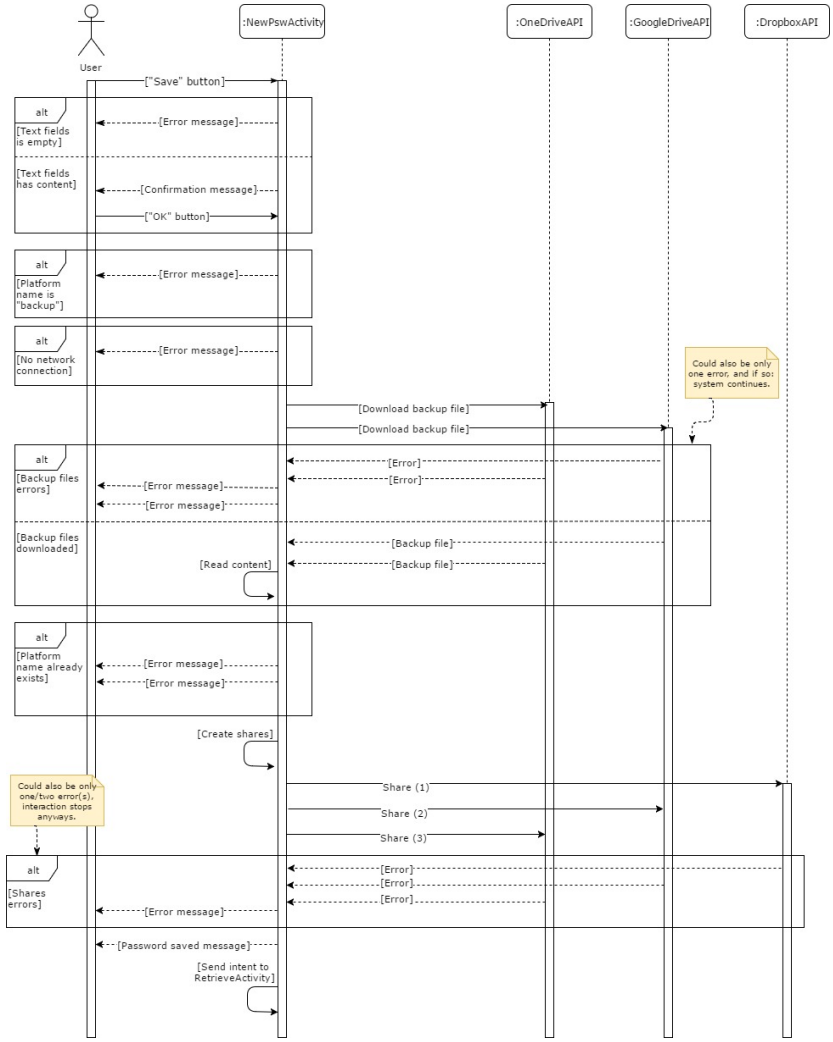

Figure 4.8: Sequence diagram for *NewPswActivity.java* file.

4.4.6 Retrieving Password

The *RetrieveActivity* is the activity for retrieving a created password, and is shown in Chapter 3, Section 3.2.4. This page is shown if the item "Retrieve passwords" in the drop-down menu is chosen, or if a new password is created. Figure 4.9 shows the sequence diagram for the retrieval function in the activity. This does not include all details, e.g. searching for shares, but is included to give an understanding of the interaction between the entities.

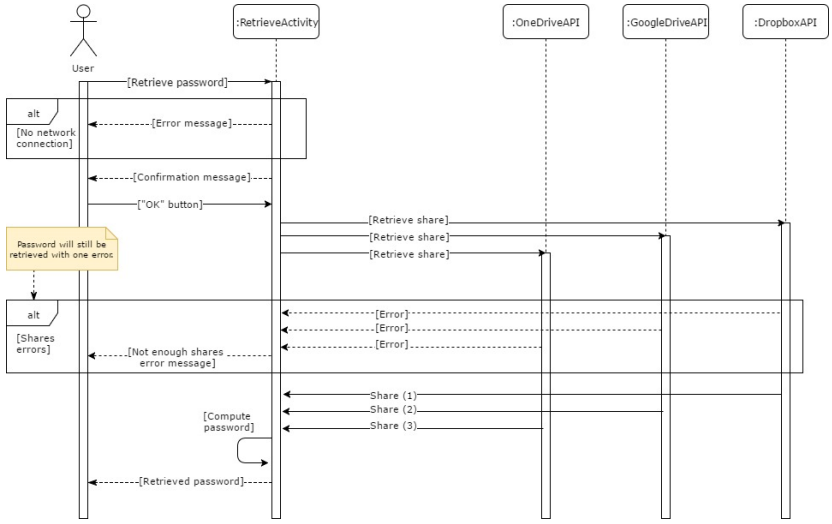

Figure 4.9: Sequence diagram for the retrieval functionality in *RetrieveActivity.java* file.

Figure 4.10 shows the workflow for this activity. Because of the amount of code in this activity, the diagram is severely simplified. Details like messages for the user and checking of network connectivity is left out, as the main purpose of the diagram is to show how the components are linked together. The functionality for retrieving passwords is the most detailed part, as this is the most important part of this activity.

The activity is started by an intent, and if the intent contains data from *NewPswActivity* - this occurs when a new password is created - the data are saved to the ListView, **lv**, containing all passwords creating. The backup files are also updated. If there are no data in the intent, the activity will display the page for retrieving and deleting passwords, as well as for manually downloading and uploading backup files. If the user clicks on **backup** or **download** for downloading and uploading the backup files, the action is performed, and the user will get a feedback indicating if it was successful or not. Also, the user may click and hold on a password in the password list to delete the specific password. When a user wants to retrieve a password, the application will attempt to retrieve the shares from the clouds, and if two or more shares is received, the password is recomputed and shown to the user.

Similar to Section 4.4.4, the diagram contains the menu options for the different scenarios in regards to how many clouds that are connected.

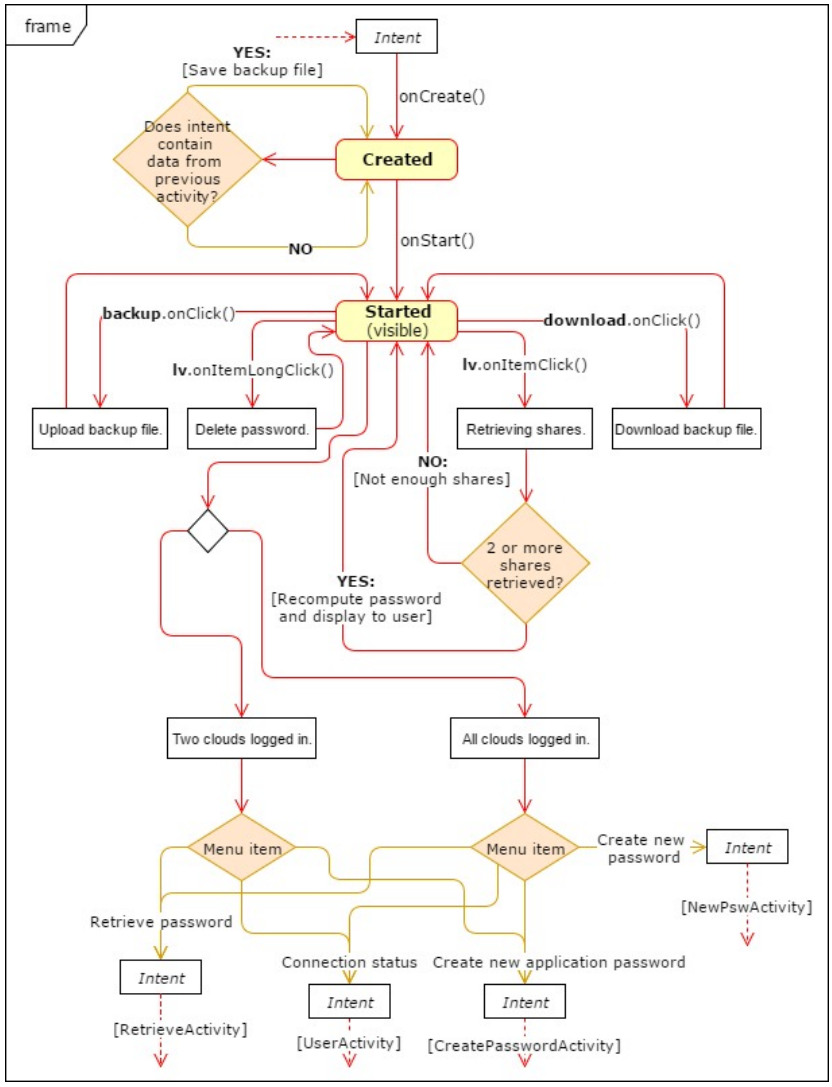

Figure 4.10: Activity diagram for *RetrieveActivity.java* file.

4.5 Secret Sharing

As described in Section 1.2 in Chapter 1, the concept of perfect threshold secret sharing is dividing data D into n pieces in such a way that D is easily reconstructable from any k or more pieces, but even the complete knowledge of *k-1* pieces reveals absolutely no information about D, as all possible values are equally likely. This is called a *(k,n) threshold scheme*.

For the application, each created password is divided into **three** pieces, where **two** or more is needed to reconstruct the password. This is known as a *(2,3) threshold scheme*. For implementing the algorithm, a repository available on GitHub is used as a base [Tie14].

4.5.1 Creating the Pieces

Listing 4.8 shows how the password, n, and k are the parameters of a method called *splitSecretIntoPieces*, where the return value is a string array containing the pieces, or shares, that will later be distributed to the clouds.

Listing 4.8: Splitting password into shares, code from *UploadSharesTask.java* file.

```
// Split password into shares.
    final String secret;
    secret = [THE PASSWORD];
    final int n = 3, k = 2;
    String[] pieces = splitSecretIntoPieces(secret, n, k);
```

The method will convert the password, or secret, into an integer, before creating a modulus by receiving the prime used for a 384-bit payload, meaning that a user can save a password that is up to 48 characters long. The application will then split the password into shares by creating coefficients, randomizing these and setting the polynomial, where the first coefficient is the secret. The polynomial is then used to produce the shares, which is then returned and all the components are placed together to a piece in the form of *n:k:x:modulus:share*. Here, x is the x in f(x) and is unique for each of the pieces made from the specific password. An example of a piece is shown in the box below.

```
3:2:1:830856716641269388050926147210378437077636615999889742043367411719044426226024000990720638469358465237775344863952732839894
2540
```

4.5.2 Reconstructing the Password

When reconstructing the password, all the collected pieces is added to an ArrayList, shuffled and added to a string array. The string array is used as a parameter for the method *mergePiecesIntoSecret*. Here, the shares are combined, and the password is reconstructed by solving the polynomial equation and finding the secret.

Listing 4.9 shows a scenario where the password is reconstructed when all pieces are collected. If only two pieces are available, the password will still be reconstructed by sending these to the method as well. By only having one piece, the user will receive an error message.

Listing 4.9: Retrieving password, code from *RetrieveActivity.java* file.

```
String[] pieces = new String[3];
pieces[0] = [Piece from OneDrive];
pieces[1] = [Piece from Dropbox];
pieces[2] = [Piece from Google Drive];

// Create arraylist out of the formatted strings,
// shuffle and reconstruct secret.

      List<String> list = new
          ArrayList<String>(Arrays.asList(pieces));
      Collections.shuffle(list);
      String[] kPieces = list.toArray(new String[0]);
      final String reconstructed = mergePiecesIntoSecret(kPieces);
```

Chapter 5
Evaluation and Results

This chapter presents the evaluation of the application, based on user tests performed by test subjects. Included are the plan and hypothesis for the evaluation, setup, and also the results drawn from the assessment.

5.1 Plan and Hypothesis

The evaluation goal was to measure how well the application performed while being used by regular users. The users were able, and encouraged, to comment on the application's UI and behavior. The aim was to have a focus group of 5 subjects, preferably with various degrees of technical skills. As the main focus was to test whether the application functions under normal usage, as well as receiving inputs from users regarding the UI, a group of this size served the purpose.

The subjects were given a test sheet containing 70 steps that covered all aspects of the functionality, including e.g. logging in, creating passwords, and verifying error messages. The users gave each functionality a score of "PASSED" or "FAILED," where "PASSED" was provided if the application responded as described in the test sheet, and "FAILED" was provided if there were any deviations. Table 5.1 shows an example of how a user may have filled out the test sheet, and the full sheets are found in Appendix A. By evaluating these tests, a quantitative score, the "overall score," was used to describe how well the application performs regarding functionality. The comments provided by the users served as inputs for further development and improvement of the application.

5.1.1 Hypothesis

By having 5 test subjects following the 70 steps, which are weighed equally, there will be 350 chances of a "PASSED" or "FAILED" functionality. This means that one failed functionality will decrease the overall score by

$$100 - \frac{349}{350} \approx 0,3\%. \tag{5.1}$$

Assuming that each subject will get at least four deviations, 20 out of the 350 chances will be marked as "FAILED." This would give an overall score of

$$\frac{330}{350} \approx 94\%. \tag{5.2}$$

As the errors found in the application will be adjusted throughout the testing period, it is reasonable to believe that the number of fails will slightly decrease over the course of the period. One can not predict how significantly the errors found will adjust the outcome of the user tests, as it would require time to fix the errors and the frequency of test completion is unknown. By assuming that three out of the five subjects, 60%, will only have two deviations, the overall score is given by

$$\frac{336}{350} = 96\%. \tag{5.3}$$

Since there are unknown factors to how well the found errors will improve the outcome, it is more likely that less than 60% of the subjects will experience a decrease of two errors, than more. By this, the null hypothesis claims that the application will have a "PASSED" percentage of 96 percent or less, and the alternative hypothesis claims that the PASSED percentage is higher than 96 percent.

H_0: The application will have a "PASSED" percentage of 96 percent or less.
H_1: The application will have a "PASSED" percentage of higher than 96 percent.

Functionality	Result	Comment
Internet Connectivity		
Turn of the Internet on your device		
Create a new password, verify error message		
Upload backup file, verify error message		No error message.
Download backup file, verify error message		
Verify no access to menu on Connect Status page		

Table 5.1: Example on how the test form cloud be filled out by a user, where a green box equals "PASSED" and a red box equals "FAILED".

5.2 Setup

The evaluation process will include the following setup:

- Test sheets created in Microsoft Excel.
- Focus group containing 5 subjects
- A LG Nexus 5X phone
- The "SecretSharing" application

The subjects were recruited from NTNU, Campus Gløshaugen, as the setup needed to perform the test were located here. This narrow area limited the distribution regarding technical skills, but as the application is new, the gender and age are distributed, and most of the test subjects are most familiar with Apple Inc.'s iOS (see Table 5.2), one can argue that the subjects represent normal users.

Subject	Technical Background	Perferred Mobile OS
1	Telematics	iOS
2	Communication Technology	iOS
3	Telematics	iOS
4	Communication Technology	Android OS
5	Energy and Environmental Engineering	iOS

Table 5.2: The subjects where recruited from NTNU, Campus Gløshaugen.

5.3 Results

Appendix B shows the complete overview of the results of the user tests, including the feedback given in the form of comments. Most of the errors found have been solved, but it has not been possible to recreate the situations where the random errors occurred. Thorough error probing was performed, and bugs that might have caused the errors were fixed. Some of the feedback are also added as future work and are discussed in Chapter 7.

The overall feedback on the application included comments stating that the application looks good and works well, but also comments stating that it is not as intuitive as it could have been.

Figure 5.1 shows the overall score of the user test, with a "PASSED" score of **98,2%**. Based on these results, the null hypothesis from Section 5.1 is discarded and concludes with the alternative hypothesis H_1 being supported.

Figure 5.1: The overall score of the user tests: "PASSED" score being 98,2% and "FAILED" score being 1,8%.

Chapter 6 Discussion

This chapter discusses the process of developing the application and explains the choices that have been made.

6.1 Development Process

The reason for developing the application for Android, instead of iOS, was mainly because of Android being Java-based and Android devices were available for testing the application, as well as it being preferred that developers use a Mac when developing for iOS. Also, as mentioned in Chapter 2, Section 2.1, Android has - as for April 2017 - approximately 65% of the marked share globally [Net].

Before choosing Android Studio as the IDE for developing the application, *Eclipse for Android Developers* and *Xamarin* were also considered [Fou17, Xam17]. Based on personal preference for developing in Java, Xamarin - which uses C# - was ruled out after testing. The Eclipse IDE was prone to errors and the code did not run as expected. This may, of course, be due to limited experience with mobile application development, but Android Studio was found to be more intuitive and easier to work with, and it is the official IDE for Android.

In the early stages of making connection to the APIs, an integration solution named *CloudRail* was used [Clo17]. CloudRail bundles the APIs into a single unified API, so instead of connecting and communicating with the individual APIs, the action was sent to CloudRail who managed the connections. The integration worked well, but the free version does not provide direct authentication - the users would receive a "Powered by CloudRail" page when authenticating. As this was not preferable, it was decided to connect to the APIs individually. Figure 6.1 shows an illustration of how CloudRail is implemented.

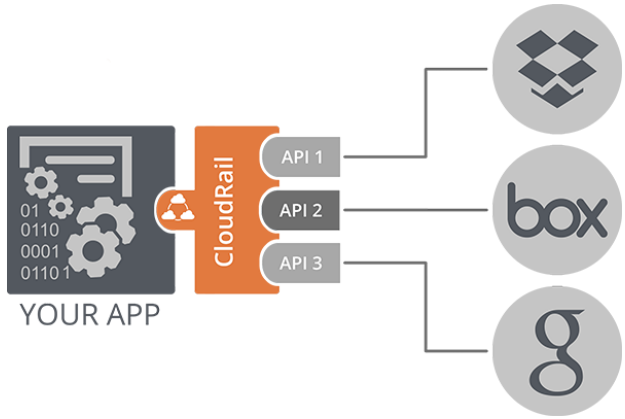

Figure 6.1: Implementation of CloudRail [Clo17].

One important part of the development was to study external libraries. SCAPI, an open-source Java library for implementing secure two-party and multiparty computation protocols, was thought to be a possibility when implementing secret sharing [SCA14]. Unfortunately, secret sharing has yet to be implemented in this library [EFLL13].

The GitHub repository used for implementing secret sharing included several methods for calculating prime numbers used to generate the shares [Tie14]. For the application, the number used, *8308567166412693 8805092614721037843700776366159998897420433674117190444262260240009907206384693584652377753448639527* is tested to be a prime number [Num17]. The method is called *getPrimeUsedFor384bitSecretPayload()*, and 384 bits equals 48 characters. By this, the password can be up to 48 characters long, which should be sufficient. During testing, all special characters added to a password has been successfully returned when reconstructing the password, including e.g. <, ®, },], and spaces.

The Google Drive API caches metadata locally, meaning that if a file gets deleted on the cloud, it will take some time for the application to register this. This does not mean that the deleted share can be retrieved and used for reconstructing the password. Listing 6.1 shows how to request a synchronization with Google using the *requestSync* method [Goo17h]. The method was implemented, but it appeared to be time-consuming, and the requests are rate limited, meaning that they cannot be performed as often as one might want. The solution was then to wait until the is automatically re-synced by Google, which has not created any problems for the functionality. It has also been noticed that

OneDrive has a tendency to use more time to connect than the other clouds. This does not affect the functionality to a great extent, as the user is asked to try again, but it is something that could be optimized.

Listing 6.1: Requesting synchronization with Google Drive

```
Drive.DriveApi.requestSync(mGoogleApiClient).setResultCallback(new
    ResultCallback<Status>() {
       @Override
       public void onResult(@NonNull Status status) {
           Log.e("sync_status", status.toString());
           if (status.getStatus().isSuccess()) {
               //Do something
           }
       }
   });
```

The reason for not implementing a (3,4) threshold scheme, as presented in the initial prototype in Chapter 3, is mainly because of time restrictions and priorities. It was found to be more important to have three working connections that demonstrates the concept of having one cloud disconnected and still being able to retrieve the password, rather than having four connections that might be prone to errors. Only two of the clouds, OneDrive and Google Drive, contains the backup files. This decision was made, similar to the implementation of clouds, because of time restrictions and priorities. It was found more important to focus on demonstrating the concept of a backup file, while also having an extra backup file in case one of them is unavailable. The implementation of more clouds and backup files are presented as future work in Chapter 7.

6.2 Considerations

To use the application, the user must have, or be willing to create, accounts on the given clouds. This might be a limitation, as users may have preferences on which clouds they want to use. Also, when a password is saved in the application, the user is responsible for the strength of the password. However, as most of the passwords are created externally before being stored in the application, it is likely to believe that the platforms have password restrictions set for the password. Also, if the user changes the password of one of the clouds externally, this will not affect the functionality of the application.

As the shares are saved on clouds that are intended for storing data available for modification, the owners of the clouds have full read/write access to the files on the clouds. This is a vulnerability of the application, as it means that an inattentive user might accidentally delete or modify the shares. VSS, described in Chapter 1, Section 1.2 could provide a solution for verifying the validity of a share.

If the phone is rebooted, the application password will remain and the clouds will stay logged in. However, if the application data is cleared, or the application is re-installed, the application password and the saved passwords will get removed, and the clouds will get logged out. This means that an attacker can re-install the application or delete all the data from it, and then create a new password for the application. However, since the clouds also will get logged out in such a scenario, the attacker will not get access to the passwords. If the phone is logged into the user's Google account, then the attacker might be able to log into Google Drive, but the other clouds would require the user to log in after logout. There is a possibility of an attacker getting hands on the user's login credentials externally, but this is out of scope for this thesis.

When a user creates the application password, this password is hashed using SHA-256, and stored in a key-value pair. When the user tries to log into the phone afterward, the input is then hashed and compared with the saved hash. Since there is no salt added to the hash function, an attacker could perform a brute force attack or use rainbow tables to get access to the application. However, this feature is mainly implemented prevent the password from being exposed if an attacker gets access to the key-value pair. It is likely to believe that a user will notice if the phone is stolen or lost, and would then delete all the shares from the clouds - and revoke the access to the application from the cloud providers manually.

If a new user is to use the application on a phone where the application is already in use, it is recommended to upload the backup file to the clouds and clear all the data from the application. Then, when the old user is to use the application again, a new application password must be set, and the backup files must be downloaded.

When a new password is created, only one backup file is needed to download the password, as the content is assumed to be identical and it contributes to being more user-friendly. However, this can lead to creating passwords that already exist. For this scenario to occur, the upload of a backup file must fail. This will return an error message to

the user, and while ignoring this, the user creates a new password. If the download of the backup file that *did not* get updated earlier is the only one that succeeds, and the other cloud is still connected - as the upload process will stop if one of the clouds is disconnected - then there is a chance of creating a password that already exists. This also requires that the user has not successfully manually uploaded the backup files or deleted a password in between, as the backup files will get updated in such cases. Also, for there to be a problem, the user must create a password with the same platform name as the one that did not get saved to the backup file on one of the clouds.

Another thing to consider is *file carving*. File carving is the method of reconstructing data from a storage, without assistance from metadata indicators or the file system that created the file [Ins13]. After the shares are deleted in the cloud, an attacker with access to the cloud might successfully reassemble the file. Also, as the Dropbox shares get saved locally before being uploaded, and is deleted afterward, there is a possibility of an attacker carving for the file [EPI13].

Chapter 7
Conclusion and Future Work

This chapter contains the conclusion of this thesis, as well as proposals for future work.

In this thesis, the development of a password storage mobile application for Android has been presented. The mobile application implements secret sharing for confidentiality and uses cloud storage services for storing the shares. A thorough technical background on Android development and integration with the cloud storage APIs was presented in Chapter 2.

In Chapter 3, the application UI was presented. This included the initial prototype explaining the basic idea of the interface, as well as the final result. Screenshots of the final UI were provided to give the reader an understanding of the application, before introducing the functionality in Chapter 4. Chapter 4 showed the elements of the application, with a focus on the activities. UML diagrams were presented to describe the system, with activity diagrams to explain the behavior, and two sequence diagrams to show the interaction between the application and the cloud APIs. The secret sharing implementation for creating the shares and recomputing the passwords was also presented.

For testing the application, user tests were conducted. In Chapter 5, the plan and hypothesizes for the testing were presented, and the setup for the testing environment was given. The result was a "PASSED" score of 98,2%, which indicated that the functionality performed better than expected and the alternative hypothesis H_1 was supported. The feedback from the test subjects stated that the application looked good and worked well, but some of the solutions could have been optimized in regards to the UI. The testing scheme used for these tests is found in Appendix A, and the results from these were shown by a pie chart using the test results presented in Appendix B.

7.1 Future Work

The application presented in this thesis could be improved by several means. If the application were to be published, one important improvement would be to implement more cloud storage services, so that a user can choose to use the clouds in which they feel most comfortable with. The user could then chose only to use e.g. three of the clouds and have the same functionality as shown in this thesis, or connect to more of them and thus improve the security by having more shares needed to be recovered. Furthermore, backup files could be saved to all clouds. As discussed in Chapter 6, the backup files are only saved to two of the clouds due to time restrictions and priorities, as the focus was on demonstrating the concept. Also, the only cloud having a dedicated folder for the shares is Dropbox. This could be implemented on the other clouds as well.

The feedback from the user tests included ideas about future work, e.g. the possibility of adding a link to the main page in the "SecretSharing" header text. Also, the idea of moving the login buttons to the connection status indicators - as a subject commented that it was found to be more intuitive to look at the middle of the screen first, than at the top. Some subjects did not find it intuitive to locate the button for uploading the backup files manually. The functionality for manually handling the backup files could be added as a separate activity, or page, in the drop-down menu. Also, the application is not compatible with OneDrive Business. This is something that should be implemented before publishing the application and would require adding an extra authentication method for OneDrive.

It could also be beneficial to add the possibility of deleting several passwords at once and to implement the functionality of restoring in case a user accidentally deletes a password. Also, when creating a new password, a second field could be implemented to verify the password before saving it.

Regarding improvement of the security, one improvement would be to give the shares random names, instead of the platform names. Also, the backup files could be shared across the clouds, similar to splitting the password. Then one would need a given number of shares to retrieve the file. However, as the platform names are already shown in the titles of the files, the content of the backup files is not considered to be sensitive. Also, when a user creates a new application password, the old password should be provided by the user.

References

[BWS+00] M.W. Bigrigg, J.J. Wylie, J.D. Strunk, G.R. Ganger, H. Kiliççöte, and P.K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, August 2000.

[Cas] M. Casserly. The best cloud storage services. `http://www.pcadvisor.co.uk/test-centre/internet/best-cloud-storage-services-2017-uk-3614269/`. Accessed: 2017-05-06.

[Clo17] CloudRail. CloudRail. `https://cloudrail.com/`, 2017. Accessed: 2017-06-15.

[Cra16] CrazyAppDev. Convenient password manager (unreleased). `https://play.google.com/store/apps/details?id=disco.ethz.ch.convenientpasswordmanager&hl=no`, 2016. Accessed: 2017-06-17.

[DH12] Ed. D. Hardt. [RFC6749] The OAuth 2.0 Authorization Framework. `https://tools.ietf.org/html/rfc6749`, 2012. Accessed: 2017-06-06.

[Dro] Dropbox, Inc. Developer branding guide. `https://www.dropbox.com/developers/reference/branding-guide`. Accessed: 2017-05-08.

[Dro17a] Dropbox. Class DbxClientV2. `https://dropbox.github.io/dropbox-sdk-java/api-docs/v2.1.x/com/dropbox/core/v2/DbxClientV2.html`, 2017. Accessed: 2017-06-06.

[Dro17b] Dropbox. Dropbox Core SDK for Java 6+. `https://github.com/dropbox/dropbox-sdk-java`, 2017. Accessed: 2017-06-08.

[Dro17c] Dropbox. OAuth guide. `https://www.dropbox.com/developers/reference/oauth-guide`, 2017. Accessed: 2017-06-06.

[EFLL13] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell. SCAPI: The Secure Computation Application Programming Interface. *Bar-Ilan University*, November 2013.

[EPI13] M. EPIFANI. Cloud storage forensics. https://www.sans.org/summit-archives/file/summit-archive-1493920922.pdf, 2013. Accessed: 2017-06-16.

[Fit16] Jason Fitzpatrick. Password Managers Compared: LastPass vs KeePass vs Dashlane vs 1Password. https://www.howtogeek.com/240255/password-managers-compared-lastpass-vs-keepass-vs-dashlane-vs-1password/, 2016. Accessed: 2017-06-07.

[Fou17] The Eclipse Foundation. Eclipse for Android Developers. http://www.eclipse.org/downloads/packages/eclipse-android-developers/neonm6, 2017. Accessed: 2017-06-15.

[Gooa] Google. Understanding Android. https://www.android.com/everyone/facts/. Accessed: 2017-05-04.

[Goob] Google. Use the Drive Badge and Brand. https://developers.google.com/drive/v3/web/branding. Accessed: 2017-05-08.

[Goo17a] Google. Accessing Google APIs. https://developers.google.com/android/guides/api-client, 2017. Accessed: 2017-06-14.

[Goo17b] Google. Android Studio - The Official IDE for Android. https://developer.android.com/studio/index.html, 2017. Accessed: 2017-06-14.

[Goo17c] Google. Application Fundamentals. https://developer.android.com/guide/components/fundamentals.html, 2017. Accessed: 2017-06-07.

[Goo17d] Google. Authenticating Your Client. https://developers.google.com/android/guides/client-auth, 2017. Accessed: 2017-06-06.

[Goo17e] Google. Authorizing Android Apps. https://developers.google.com/drive/android/auth, 2017. Accessed: 2017-06-08.

[Goo17f] Google. Content Providers. https://developer.android.com/guide/topics/providers/content-providers.html, 2017. Accessed: 2017-06-07.

[Goo17g] Google. Declaring Permissions.
https://developer.android.com/training/permissions/declaring.html, 2017.
Accessed: 2017-06-14.

[Goo17h] Google. DriveApi.
https://developers.google.com/android/reference/com/google/android/gms/drive/DriveApi.html, 2017.
Accessed: 2017-06-16.

[Goo17i] Google. Encryption.
https://source.android.com/security/encryption/, 2017.
Accessed: 2017-06-14.

[Goo17j] Google. Introduction to Activities.
https://developer.android.com/guide/components/activities/intro-activities.html, 2017.
Accessed: 2017-06-07.

[Goo17k] Google. Introduction to the Google Drive Android API.
https://developers.google.com/drive/android/intro, 2017.
Accessed: 2017-06-05.

[Goo17l] Google. Manifest.permission.
https://developer.android.com/reference/android/Manifest.permission.html, 2017.
Accessed: 2017-06-12.

[Goo17m] Google. The Activity Lifecycle.
https://developer.android.com/guide/components/activities/activity-lifecycle.html, 2017.
Accessed: 2017-06-07.

[Ins13] InfoSec Institute. File Carving.
http://resources.infosecinstitute.com/file-carving/#gref, 2013.
Accessed: 2017-06-16.

[KSBK15] R. Koppela, S. Smith, J. Blythe, and V. Kothari. Workarounds to Computer Access in Healthcare Organizations: You Want My Password or a Dead Patient? *University of Pennsylvania, Dartmouth College and University of Southern California*, 2015.

[Li16] J. Li. Lecture notes from the course TDT4237 - Software Security: Mobile Application Security. *The Norwegian University of Science and Technology*, 2016.
Accessed: 2017-06-14.

[Mic] Microsoft. Branding guidelines.
https://msdn.microsoft.com/en-us/onedrive/dn673556.aspx.
Accessed: 2017-05-08.

[Mic15] Microsoft. OneDrive authentication and sign-in.
https://dev.onedrive.com/auth/msa_oauth.htm#authentication-scopes, 2015. Accessed: 2017-06-06.

[Mic17] Microsoft. OneDrive SDK for Android. https://github.com/OneDrive/onedrive-sdk-android, 2017. Accessed: 2017-06-12.

[Net] Netmarkedshare. Mobile/Tablet Operating System Market Share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1. Accessed: 2017-05-04.

[Nie95] J. Nielsen. 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/, 1995. Accessed: 2017-05-09.

[Nor] A. Nordrum. Quantum Computer Comes Closer to Cracking RSA Encryption. http://spectrum.ieee.org/tech-talk/computing/hardware/encryptionbusting-quantum-computer-practices-factoring-in-scalable-fiveatom-experiment. Accessed: 2017-05-05.

[Num17] Numberempire. Prime Numbers Generator and Checker. http://www.numberempire.com/primenumbers.php, 2017. Accessed: 2017-06-15.

[SB05] A. Subbiah and D.M. Blough. An approach for fault tolerant and secure data storage in collaborative work environments. *Storagess '05: Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, pages 84–93, November 2005.

[SCA14] SCAPI. Welcome to SCAPI. https://scapi.readthedocs.io/en/latest/, 2014. Accessed: 2017-06-14.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[Sti92] D.R. Stinson. An Explication of Secret Sharing Schemes. *Designs, Codes and Cryptography*, 2(4):357–390, May 1992.

[Sun] Sungard Availability Services. Top Three IT Workplace Issues Prevent CIOs from Sleeping Easy in 2015. http://www.sungardas.com/en/about/news/2015/january/top-three-it-workplace-issues-prevent-cios-from-sleeping-easy-in-2015/. Accessed: 2017-05-06.

[Tie14] T. Tiemens. Shamir's Secret Share in Java. https://github.com/timtiemens/secretshare, 2014. Accessed: 2017-06-08.

[Vls] Vlsergey. 3 polynomials of degree 2 through 2 points, Wikimedia Commons (CC-BY 3.0). https://commons.wikimedia.org/wiki/File:3_polynomials_of_degree_2_through_2_points.svg. Accessed: 2017-05-05.

[Win] D. Winder. How secure are Dropbox, Microsoft OneDrive, Google Drive and Apple iCloud cloud storage services? http://www.alphr.com/apple/1000326/how-secure-are-dropbox-

microsoft-onedrive-google-drive-and-apple-icloud-cloud-storage.
Accessed: 2017-05-06.

[Xam17] Xamarin. Xamarin.
https://www.xamarin.com/, 2017.
Accessed: 2017-06-16.

Appendix A
User Test Sheet

Functionality	Result	Comment
PART ONE - Basic Functionality		
Application Log-in		
Open the application		
Fill in two different passwords on creating password page, verify error message		
Create a password		
Log in with a wrong password, verify error message		
Log in with the correct password		
Cloud Services Log-in		
Approve storage permissions		
Log in to Dropbox		
Log in to One Drive		
Log in to Google Drive		
Internet Connectivity		
Turn off the Internet on your device		
Create new password, verify error message		
Upload backup file, verify error message		
Download backup file, verify error message		
Verify no access to menu on Connect Status page		
Verify disconnected clouds on Connect Status page		
Verify error message on Connect Status page		
Verify no access to buttons for cloud login		
Turn on the Internet on your device, and reload application		
Verify access to menu on Connect Status page		
Veriy connected clouds on Connect Status page		
Verify connected message on Connect Status page		

Figure A.1: User test: PART ONE - Basic Functionality (1).

Create and Retrieve a New Password		
Fill in only platform field, verify error message		
Fill in only password field, verify error message		
Create password with the platform name "BACKUP", verify error message		
Create a new password		
Verify "Backup downloaded"/"No backup file found"		
Verify new page successfully loaded		
Verify correct passwords in list, including those from the backup list		
Verify "Backup updated on Google Drive" and "Backup updated on One Drive"		
Retrive the new password by click		
Delete Password		
Long click on password, verify warning message, press OK		
[EXTERNAL] Verify password deleted and backup file updated on clouds		
Delete all passwords, verify backup file deleted		
Download backup file, verify "No backup file found"		
Upload backup file, verify error message		
Upload and Download Backup File		
[EXTERNAL] Create a new password, verify backup file updated on One Drive and Google Drive		
Upload backup file, verify warning message, click OK		
[EXTERNAL] Verify no changes in backup files		
Download backup file and reload page, verify no changes		
Create New Application Password		
Change password for application and log in		
Log Out From the Clouds		
Log out from the clouds		

Figure A.2: User test: PART ONE - Basic Functionality (2).

Functionality	Result	Comment
PART TWO - Advanced Functionality		
Clear data from app in "Settings" and log in again		
Downloading Backup File		
Log in to clouds		
[EXTERNAL] Create password with a platform name from the external backup file, verify error message		
Create a new password		
[EXTERNAL] Verify backup updated on One Drive and Google Drive		
Delete password		
[EXTERNAL] Verify backup updated on One Drive and Google Drive		
Log out from clouds		
Testing: One Drive Disconnected		
Log in to Google Drive		
Log in to Dropbox		
In menu: verify access to retrieving passwords, no access to making new passwords		
Download backup file, verify no change in password list		
Retrive password		
Delete password, verify error message		
Log out from clouds		
Testing: Dropbox Disconnected		
Log in to Google Drive		
Log in to One Drive		
Retrive password		
Delete password, verify error message		
Log out from clouds		
Testing: Google Drive Disconnected		
Log in to OneDrive		
Log in to Dropbox		
Download backup file, verify no change in password list		
Retrive password		
Delete password, verify error message		
Log out from clouds		
[EXTERNAL] Delete all login info from the phone		

Figure A.3: User test: PART TWO - Advanced Functionality.

Appendix B
User Test Results

Functionality	PASSED	FAILED	Comments
PART ONE - Basic Functionality			
Application Log-in			
Open the application	5		*General: maybe implement a link to home page on the top "SecretSharing" text [Future Work]*
Fill in two different passwords on creating password page, verify error message	5		*" ✓ in keyboad does not automatically submit request [SOLVED]*
Create a password	5		*Maybe popup with "password successfully created" [ADDED]*
Log in with a wrong password, verify error message	5		
Log in with the correct password	5		
Cloud Services Log-in			
Approve storage permissions	5		
Log in to Dropbox	5		*Could maybe add login to the status symboles [Future Work]*
Log in to One Drive	4	1	*Not compatible for One Drive Business [Future Work]*
Log in to Google Drive	5		*Lost Internet connection after having logged in.*
Internet Connectivity			
Turn off the Internet on your device			
Create new password, verify error message	5		
Upload backup file, verify error message	5		*Not intuitive where to find "upload backup file" [Future Work]*
Download backup file, verify error message	5		
Verify no access to menu on Connect Status page	5		
Verify disconnected clouds on Connect Status page	5		
Verify error message on Connect Status page	5		
Verify no access to buttons for cloud login	5		
Turn on the Internet on your device, and reload application			
Verify access to menu on Connect Status page	5		
Verify connected clouds on Connect Status page	5		
Verify connected message on Connect Status page	5		
Create and Retrieve a New Password			
Fill in only platform field, verify error message	5		*Not sure what "platform" means. [ADDED "e.g. "Facebook"" TO HINT]*
Fill in only password field, verify error message	5		
Create password with the platform name "BACKUP", verify error message	4	1	*"BaCkUp" did not give error message [SOLVED]*
Create a new password	5		
Verify "Backup downloaded"/"No backup file found"	5		
Verify new page successfully loaded	5		
Verify correct passwords in list, including those from the backup list	5		
Verify "Backup updated on Google Drive" and "Backup updated on One Drive"	4	1	*IOException: Not able to create backup file, random error [SEMI-SOLVED]*
Retrive the new password by click	5		
Delete Password			
Long click on password, verify warning message, press OK	5		
[EXTERNAL] Verify password deleted and backup file updated	4	1	*Not able to delete since it does not exist. Manually upload backup file to update.*
Delete all passwords, verify backup file deleted	5		
Download backup file, verify error message	5		
Upload backup file, verify error message	5		
Upload and Download Backup File			
[EXTERNAL] Create a new password, verify backup file updated on One Drive and Google Drive	4	1	*IOException: Not able to create password whit no backup file, random error [SEMI-SOLVED]*
Upload backup file, verify warning message, click OK	5		
[EXTERNAL] Verify no changes in backup files	5		
Download backup file and reload page, verify no changes	5		
Create New Application Password			
Change password for application and log in	5		*No password requirements? [ADDED PASSWORD STRENGTH INDICATOR]*
Log Out From the Clouds			
Log out from the clouds	5		
PART TWO - Advanced Functionality			
Clear data from app in "Settings" and log in again			*Not sure what that meant, thought it meant to delete all passwords. [Added "in Settings"]*
Downloading Backup File			
Log in to clouds	5		
[EXTERNAL] Create password with a platform name from the external backup file, verify error message	5		
Create a new password	5		
[EXTERNAL] Verify backup updated on One Drive and Google Drive	5		
Delete password	5		
[EXTERNAL] Verify backup updated on One Drive and Google Drive	5		
Log out from clouds	5		
Testing: One Drive Disconnected			
Log in to Google Drive	5		
Log in to Dropbox	5		
In menu: verify access to retrieving passwords, no access to making new passwords	5		
Download backup file, verify no change in password list	5		
Retrive password	5		
Delete password, verify error message	5		
Log out from clouds	5		
Testing: Dropbox Disconnected			
Log in to Google Drive	5		
Log in to One Drive	5		
Retrive password	5		*Didn't have enough shares the first time, OD is slow.*
Delete password, verify error message	5		
Log out from clouds	5		
Testing: Google Drive Disconnected			
Log in to OneDrive	5		
Log in to Dropbox	5		
Download backup file, verify no change in password list	4	1	*Error: "GoogleApiClient must not be null" [SOLVED]*
Retrive password	5		
Delete password, verify error message	5		
Log out from clouds	5		
[EXTERNAL] Delete all login info from the phone	5		
TOTAL	**98,2 %**	**1,8 %**	

Figure B.1: Results from the user tests.