PassSafe: An Android Password Manager Application

Gongyin He

Master of Science
Computer Science
School of Informatics
University of Edinburgh
2017

Abstract

Since the Internet plays a growing role in humans' life, most of users own a large number of accounts. To remember the same quantity of passwords which are challenging for memory limitation, many of them choose weak passwords and reuse passwords, which would significantly decrease their password security. Therefore, password managers are introduced to help users by generating strong and unique passwords and retrieving passwords for them. This thesis analyses the current implementation schemes for password managers and explores potential design schemes for improvement. We implement PassSafe, an open-source Android password manager, which is expected to have robust security performance (e.g. an improved password generating scheme) and provide not only fundamental functions of a password manager but also some extra features to improve convenience (e.g. autofill). Finally, we evaluate the performance of PassSafe.

Acknowledgements

Many thanks to my supervisor Dr Myrto Arapinis for her patient support, excellent guidance and advice throughout the project. Also, many thanks to the participants in the usability test for the time and valuable feedback. Finally, I would like to thank my parents for supporting me to the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Gongyin He*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Text passwords are the most dominant mechanism to authenticate human users on the Internet. Users have to use text password to manage various types of accounts, such as email, cloud drive, online shopping, banks, social media and games. Password security can directly determine the security of users' property. Therefore, users are advised to use strong passwords, use unique passwords for different sites and change passwords regularly. However, these requirements are challenging for most of the users, especially when they are overwhelmed by a large number of accounts, and the number of accounts is still growing over time. As a consequence, there is a dilemma users have to face: making their account vulnerable or forgetting their passwords frequently. To remember them easily, many users prefer to choose weak passwords or reuse a password in multiple accounts. [1] studied data from over half a million users over a period of three months in 2007, and the result shows that the average user has 6.5 passwords, each of which is shared across 3.9 different sites, and the majority of users choose passwords with lower case letters if not forced to do so. [2] also observed that 43% of users directly reuse passwords between websites. Password weakness and reuse are the two most important reason for the insecurity of password because they make passwords very vulnerable to brute force attack and phishing attack.

Password manager applications aim to provide a way out of this dilemma. A password manager application could generate strong and diverse passwords, store passwords securely and retrieve passwords for users conveniently. The usability evaluation in [3] shows that users overall prefer portable password managers among a phone manager, a USB manager and an online manager, and non-technical users show a strong inclination towards the phone manager. Also, mobile devices (e.g. cell phones and tablets) have become one of the most important portals for users to access the Internet.

[4] reports that Android accounts for more than 60% of mobile/tablet operating system market share in 2017. Therefore, an Android password manager application could provide good usability, portability and accessibility for users to manage their passwords. There are more than 200 password manager apps in Google Play Store[1] now, but the most of the popular ones are closed-source (e.g. 1Password [5], LastPass [6] and DashLane [7]), and suffer many critical security vulnerabilities [8][9][10]. Consequently, PassSafe, an Android password manager application, is developed in this project, which is open-source[2] and attempts to provide better security and usability performance.

1.1 Objective

The objective of this project is to develop a practical Android password manager application which can help users to store and manage their account information (e.g. usernames and passwords) conveniently and securely with a single master password set by users. The application should also be able to fill stored passwords into other third-party applications automatically.

1.2 Dissertation Outline

This dissertation is structured as follows:

Chapter 2: Background

This chapter summarises the relevant background key to understanding the dissertation. It reviews the development of password managers and introduces the primary concepts of an Android application.

Chapter 3: Analysis and design

This chapter discusses the implement schemes of core modules and addresses the final design of them, including password generator, database, authentication and autofill.

Chapter 4: Implementation

This chapter presents PassSafe, the final implementation of Android password manager in this project. It shows user interfaces, general work flows and technical details of every primary component of PassSafe.

[1]https://play.google.com/store/search?q=password+manager&c=apps&rating=0
[2]https://github.com/JoeyHe912/PassSafe

Chapter 5: Evaluation

This chapter demonstrates the usability evaluation and the feature-based comparison for evaluating PassSafe.

Chapter 6: Conclusion

This chapter concludes the aim of this project, highlights the accomplishment and indicates the future work for improvement.

Chapter 2

Background

This chapter gives the background information of this project, which includes the password managers and Android applications. Section 2.1 reviews the development of password managers, discusses different implementation schemes and tries to conclude advantages and drawbacks of them. Section 2.2 outlines the general concepts of Android applications regarding development environment, components and compatibility.

2.1 Password Managers

Password security is concerned by a large number of researchers, and some alternative passwords are introduced attempting to improve it, such as graphical passwords and biometrics. However, they have practical issues with privacy, theft, or the huge infrastructural costs of deployment and maintenance. Because of the combination of low cost, immediacy and convenience, traditional text passwords may not be abandoned for a long time in many scenarios [11]. Therefore, instead of replacing text passwords, approaches to improve password security could focus on strengthen and support passwords themselves. Password managers are mainly designed to improve text password security by, for example, increasing password entropy, reducing password reuse and securely storing and retrieving passwords for users.

In early implementations of the concept of password manager, they attempt to raise entropy of password by stretching the original passwords. [12] uses a hash function to derive new passwords from original passwords. The new passwords have a longer length which can significantly increase the time an adversary performing brute-force attack. The primary drawback of this scheme is that it cannot reduce password reuse because the same original passwords will derive the same new passwords. Remembering a large number of original passwords is still challenging for users' memory. Site-specified password managers [13][14][15] are introduced to tackle this problem, which only require users to remember a master password. The common idea of this scheme is to generate a new password by hashing a combination of a master password and a value related to the site (e.g. domain, name or an alias defined by users). This kind of approaches has some advantages, for example, slight memory burdens for users, low costs for deployment, high usability for working on multiple platforms as no synchronisation required and resistance to some phishing attacks. However, lack of flexibility is a critical drawback of these approaches, because all generated passwords of a user are tightly related to a master password. If a user wants to change the master password, all generated passwords have to be reset; similarly, if a user wants to change one of the generated passwords, it can only be achieved by modifying the master password, and all other passwords also have to be reset.

Early widely-used practical password manager applications are mainly built in browser software (e.g. Google Chrome and Microsoft Internet Explorer) or as browser extensions. Most of them focus on storing and retrieving passwords for users. When users sign in a website, the domain, username and password can be automatically saved, and when users open the website again, the login information can be automatically filled in the form. This scheme can relieve password fatigue and reduce sign-in time. Browser-based password managers may resist naive phishing attack to some extent because they fill the passwords depending on the web domain. However, they also have many vulnerabilities to web attacks. [16] demonstrates a Cross-site Scripting (XSS) attack to steal password data provided by the password manager. [17] reports a Reverse Cross-Site Request (RCSR) vulnerability leaking passwords that many browser-based password managers may fill in passwords in different login forms from the original ones where the passwords were first filled and saved. [18] presents several security flaws of browser-based password managers and demonstrates a phishing attack to exact a user's password remotely. Another drawback of these password managers is that they cannot fundamentally improve users' password quality. Password weakness and reuse still exist and make users' credentials vulnerable.

Modern password manager applications attempt to provide more comprehensive functionalities and more secure services. [19] introduces Tapas, a dual-possession-authentication password manager. It requires users to authenticate with a desktop computer and a smartphone, which can significantly increase the security performance. However, the convenience is undermined, since users must operate two devices at the same time to use the manager. AutoPass [20] improves the flexibility of site-specified password managers by adding and storing offsets in password generation function which allows users change a password without changing the master password. It also uses the Password Requirements Markup Language (PRML) to store site password policy and user preference. [21] introduces Versipass with a new password manager paradigm, which incorporates traditional password managers and cued graphical passwords. It uses user-input image cues to generate and retrieve passwords for users. Hence users do not need to remember any text password. However, user study of Versipass shows that a new and unfamiliar password scheme (i.e. image cues) confuses many users. For current popular practical password managers (e.g. 1Password [5] and LastPass [6]), they provide rich features, such as generating random passwords, automatically filling passwords, storing encrypted password database on servers, and supporting for multiple platforms. However, most of them have critical vulnerabilities in many aspects. [9] reports four types of vulnerabilities (i.e. Bookmarklet, Web, Authorization and User Interface vulnerabilities) in 5 modern password managers including LastPass. [10] presents database format vulnerabilities in 9 password managers including 1Password which cannot resist to both two attack models defined in the paper. [8] analyses 21 popular free and paid password managers, and all of them have the clipboard vulnerability.

2.2 Android Application

Android is an open-source mobile operation system developed and maintained by Open Handset Alliance (OHA) of Google. It is designed for touchscreen mobile devices such as smartphones and tablets. The first commercial Android device was released in September 2008, and now it has become the mobile operation system with the largest market share [4].

Android applications are developed with the Android software development kit (SDK) normally in Java programming language. The integrated development environment (IDE) for Android applications used to be Eclipse using the Android Development Tools (ADT) plugin, while Google released its official IDE Android Studio based on IntelliJ IDEA in 2013, which is employed in this project. It has many useful features such as Gradle-based build support, Android-specific refactoring and quick fixes, a rich layout editor allowing drag-and-drop UI components and previewing layout, etc. An Android application is finally compiled to an Android Package (APK) which can be installed by users to extend the functionality of devices. Since the Android system is based on Linux kernel, every Android application could run in its own Linux kernel, and each process has its own virtual machine (VM), to isolate with each other, which can significantly improve its security.

There are four types app components (i.e. activities, services, broadcast receivers and content providers) as essential blocks for building an Android application, while an application does not have to contain all types of components[1]. Each type has its own distinct function and lifecycle. Activities are the primary components interacting with users, each of which is a subclass of Activity class and may represent a single screen interface. A service does not provide a user interface, which runs in the background to perform long-running operations or to perform work for remote processes. It may be started and terminated by an activity. A broadcast receiver enables an application to receive broadcast announcements from the system. Hence the app may respond to messages from the system even when the app is not running currently. A content provider allows other applications to query and modify a shared set of data in an application if permitted.

Different devices may run different versions of the Android platform, while a successive platform version often has new APIs not available in the previous version. Therefore, an Android application should be configured with the minSdkVersion and the targetSdkVersion attributes. The minSdkVersion specifies the minimum platform version which the application is compatible with. A lower version compatibility can target on more devices, but fewer features will be available. The targetSdkVersion specifies the highest version the application is optimised on, which should always target to the latest version[2].

[1]https://developer.android.com/guide/components/fundamentals.html
[2]https://developer.android.com/guide/practices/compatibility.html

Chapter 3

Analysis and Design

This chapter analyses the potential implementation schemes of several modules of the project and gives the final design of them. Section 3.1 reviews and evaluates two traditional categories of password generator implementation schemes, and finally tries to introduce an improved scheme and uses six attack scenarios to compares the security performance of three schemes. Section 3.2 describes the selection, schema design and encryption scheme of the database. Section 3.3 compares three potential authentication mechanisms. Section 3.4 discusses five schemes to implement autofill function, and finally chooses one with a better balance of security and usability performance.

3.1 Password Generator

In former password managers, password generating schemes can be divided into two categories. They are presented as scheme A and B in Figure3.1.

The scheme A can generate random passwords. It is composed of a random number generator which produces random numbers and an encoder which maps a series of number to a text password. This scheme is widely used in popular commercial password manager applications, such as LastPass [6] and 1password [5]. [22] presents three distinct approaches to implement this schema: ALPHANUM Generator, DICEWARE Generator, PRONOUNCE3 Generator. ALPHANUM Generator simply encodes random numbers to alphanumeric characters; DICEWARE Generator uses random numbers to select words from a list of common words and concatenate them as the password which is easy to remember but has relatively low entropy; PRONOUNCE3 Generator can produce pronounceable passwords by a sophisticated algorithm while the entropy of passwords will also decrease. We also viewed three open-sourced Android pass-

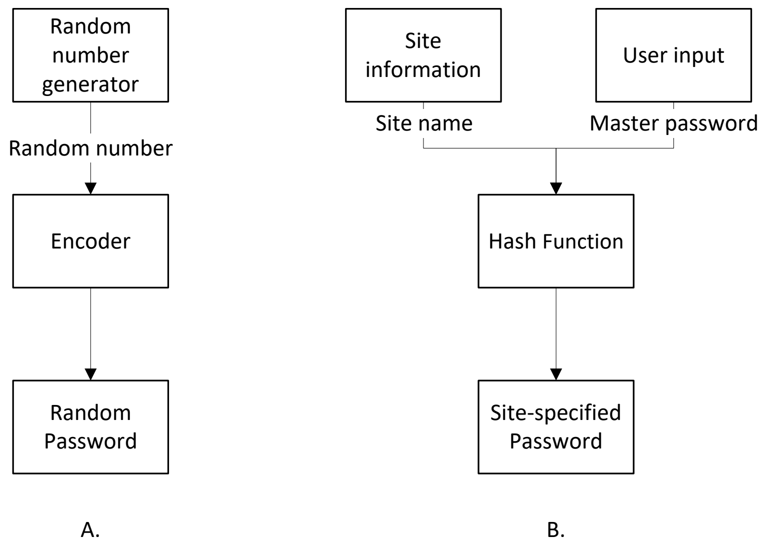


Figure 3.1: Two categories of password generating schemes

word manager, KeePassDroid [23], Puff [24] and Padlock [25]. All of them just repeat generating a random number and using the random number to select a character from a list of candidate password characters, and finally concatenate the randomly selected characters as the password.

The scheme B is to generate site-specified passwords which are hashed from site-related values and user-input values. This scheme is applied in many academic password manager papers. [13] introduces a naive procedure to produce a site-specified password by using the MD5 algorithm to hash the concatenation of a password from the user and an easy to remember the name of the website. [14] describes a browser extension, PwdHash, which can generate a different password for every site by cryptographically hashing the combination of the password entered by the user, data associated with the website, and (optionally) a private salt stored on the client machine. [15] uses a two-step method that firstly, a variable V is derived by hashing the concatenation of the username and the master password k_1 times, and secondly, the site password is derived by hashing the concatenation of the site name, the master password and the variable V k_2 times. This method improves the security, especially for defending brute-force attack.

Comparing the two schemes, the main feature of scheme A is flexibility that users can repeat the generating function to generate unique passwords for a particular site and choose a favourite password. The security of schema A depends on the quality of the random number which must be nondeterministic and nonpredictable. The random number generator used on Android is based on /dev/random or /dev/urandom, but [26] proved they are not robust due to the behaviour of the entropy estimator and the mixing function used to refresh its internal state. In addition, because it will generate different passwords every time, passwords have to be stored for retrieving. The storage of passwords could become another vulnerability for adversaries, if without proper encryption.

The main feature of scheme B is portability that no storage is necessary and users can directly use it on multiple devices without synchronising any data because the same inputs can always derive the same password. However, the feature of scheme B cause some remarkable problems. Firstly, lack of storage causes a problem that when users initially use the password manager with scheme B, they have to change their passwords currently being used to site-specified passwords generated by the manager, which may discourage users in the first step. Secondly, passwords generated from a specific site and a master password are fixed, so if one of a user's passwords need to be changed, the master password has to be changed to generate a different password, and change of master password results all other passwords have to be regenerated and reset. In addition, it is hard for users to manage two accounts on a site because they must have the same password. Finally, there is another potential risk that if one of the user's password is accidentally compromised, and the adversary knows the hash algorithm, the adversary can use a brute force attack by trying all possible master passwords. In the worst case, if the adversary hits the master password, all passwords of the user can be recovered by the adversary, which is catastrophic.

As discussion mentioned above, both of the schemes have some critical drawbacks. Therefore, we will introduce a new scheme in this project trying to solve those problems, which is shown as scheme C in Figure 3.2.

Scheme C combines ideas from both scheme A and B by hashing and encoding random numbers and the master passwords to a password. The combination of a group of random numbers and a master password is as the input of a cryptographically hash function, and then the result of hash is sent to an encoder to derive a password with a specified format. Because users may choose a master password with low entropy, a naive hash function is vulnerable to brute force attack. Hash stretching should be used to decrease the rate of attack, which is implemented by a well-known algorithm PBKDF2 [27]. The default pseudorandom function in PBKDF2 is HMAC-SHA-1, while SHA-1 collision attacks have been proven practical [28]. Therefore,

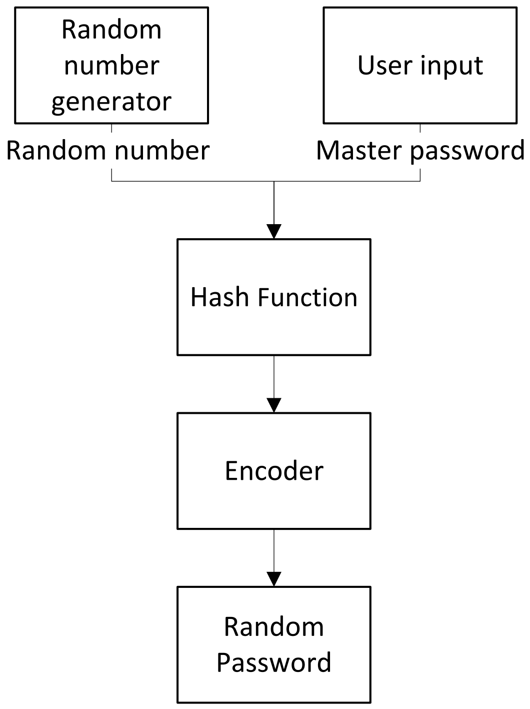


Figure 3.2: Scheme C for generating passwords

PBKDF2 with HMAC-SHA-256 is used in PassSafe, which is more secure and more time-consuming. The iteration number should be as large as possible to increase the time for computing the hash function, but if the computing time is too long, a significant delay in the application will undermine user experience. Therefore, a balance should be found with experiments on real devices. The implementation of the encoder will be introduced in section 4.7. This scheme significantly decreases vulnerabilities from random numbers compared with scheme A and provides considerable flexibility for users compared with scheme B.

To compare the security of the three schemes, six attack scenarios are defined as followed.

1. One of the **user's passwords** is compromised. For example, one of the websites stores user passwords in plain text and its database is attacked.

   In scheme B, because the master password is defined by the user, it may have relatively low entropy. The attacker can use a brute force attack by trying all possible master passwords. We can assume the attacker also knows the hash function. As a consequence, if the attacker can hit the master password, all passwords of the user can be recovered by the attacker, which is extremely dangerous. For scheme A, all user passwords are irrelevant with each other so that other passwords will be secure. For scheme C, the attacker cannot recover other passwords without the random numbers.

2. The **master password** is compromised. For example, the user's device is rooted, and when the user is typing in the master password, it may be recorded by malware.

   This scenario is similar to scenario 1. Therefore, scheme B is not secure, while scheme A and C can resist to this attack.

3. The **database** of the Password Manager is compromised. For example, the user's device is stolen, the attacker can directly access to the device.

   Scheme B does not need a database, so it is without concerns of this threat. In scheme A and C, if the database is cryptographically encrypted, the attacker will hardly recover the user passwords.

4. The **random numbers** used to generate the passwords are compromised.

   Random numbers are not used in scheme B. Scheme A only uses the random numbers to generate the passwords. Consequently, it will fail to resist this attack. In comparison, Scheme C uses the combination of the master password and the random numbers, which means the attacker will not be able to recover the user passwords with only part of them.

5. Both the **database** of the Password Manager and **master password** are compromised. (2+3)

   This scenario is a combination of scenario 2 and 3. Scheme B fails to resist scenario 2. With the master password, the attacker can decrypt the database, so the Scheme A and C fail to resist.

6. Both **random numbers** and the **master password** are compromised. (2+4)

   This scenario is a combination of scenario 2 and 4. Scheme A fails to resist scenario 4 and scheme B fails to resist scenario 2. With both random numbers and the master password, the attacker is able to regenerate the user passwords, so the scheme C fails.

As we can see in Table 3.1, scheme C can resist the most number of attack scenarios, and the two attack scenarios it cannot resist also cannot be resisted by scheme A and B. Therefore, scheme C is applied in PassSafe.

Table 3.1: Table of resisting to attack scenarios for scheme A, B and C

Attack scenario	scheme A	scheme B	scheme C
user's passwords	$\surd$	$\times$	$\surd$
master password	$\surd$	$\times$	$\surd$
Database	$\surd$	$\surd$	$\surd$
random numbers	$\times$	$\surd$	$\surd$
database + master password	$\times$	$\times$	$\times$
random numbers + master password	$\times$	$\times$	$\times$

3.2 Database and Encryption

Because the password generator scheme used in this project will generate random and unique passwords, to help users manage their passwords, the password managers should be able to store and retrieve passwords for users securely. This section will consider the storage and its security.

We considered two candidate schemes to store the users passwords. The first scheme is to store the random numbers used to generate the passwords, while the second scheme is to store the passwords with encryption. The advantage of the first scheme is that it does not require reversible encryption. Consequently, even if an adversary can obtain the database, he cannot conduct a brute-force attack to recover the master password or generated passwords. However, this project will provide a password generator which can be customised by users (e.g. setting the length and composition of passwords). Hence the custom information should also be stored in order to recover the passwords. if this information is not encrypted, it may provide a significant value to adversaries, while if this information is encrypted, this scheme will lose its advantage compared with the second scheme. Another drawback of the first scheme is that it does not allow users to store passwords created by themselves. Therefore, we will employ the second scheme.

Apart from storing every user password, users may also want to store related information, e.g. username and URL of the website. Therefore, a relational database is suitable for this circumstance. SQLite is decided to be the database management system, which is a lightweight disk-based database that does not require a separate server process and well supported in Android[1].Its library size is less than 500 KiB, but it is fully functional with SQL query statements. The database schema is designed as Figure 3.3. There is one table in the database named Password Note, which has 11 attributes. Id is the primary key. Attributes name, username, password, website, note and is_favourite are set and edited by users, while id, package_name, created_time, modified_time and accessed_time are automatically generated and maintained by the application.

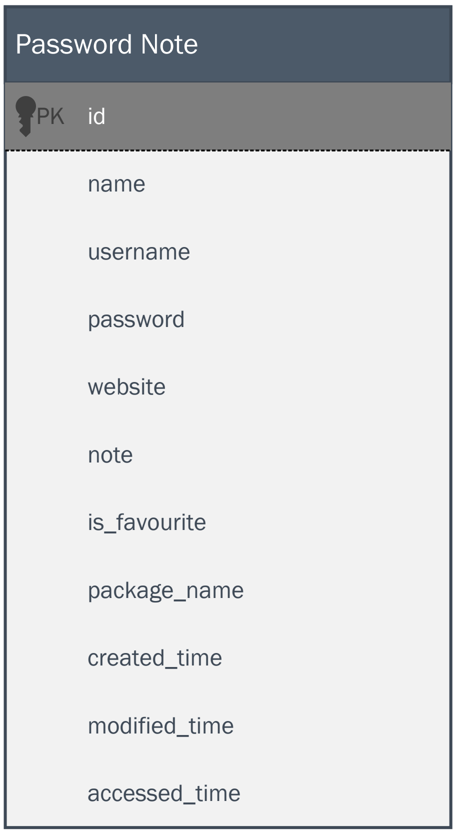


Figure 3.3: Database schema

The data on a user's disk can be easily exposed to an adversary. Hence users' passwords definitely cannot be stored in plain text and should be securely encrypted. However, if only passwords are encrypted, when the database is stolen by an adversary, the adversary can obtain websites and usernames which are also very important for users. All data stored in the database are valuable, so we decide to encrypt the whole database file.

[1]https://developer.android.com/reference/android/database/sqlite/package-summary.html

PKCS#5 [27] introduces a Password-based Encryption Standard, which is widely used in cryptographic applications. The key derivation function PBKDF2 and the Encryption scheme PBES2 are employed in PassSafe. PBKDF2 applies a pseudorandom function with a password, salt and iteration number as input to produce a key, which will be used as the encryption key. As we mentioned in section 3.1, the default pseudorandom function in PBKDF2 HMAC-SHA-1 is not safe. Hence HMAC-SHA-256 is used as the pseudorandom function. The password is the master password set by the user. Salt is a random number, which can enhance the resistance to dictionary attack and decrease the chance of collision. Iteration number defines the repeating times, which can increase the cost of exhaustive search for passwords. PBES2 will use a key derived from PBKDF2 to encrypt data. While recommended encryption scheme in the paper is DES-CBC-Pad, DES is insecure and has been superseded by AES now. Therefore, AES-CBC-Pad is used in PassSafe.

3.3 Authentication Mechanism

For generating passwords and encrypting the database, a master password should be set by users when PassSafe is initialising. The master password will also be used to authenticate the users when they are trying to open the application. Three schemes are considered as the authentication mechanism.

1) Store the master password in plain text, and then compare it with user input.
2) Try to decrypt the database with user input, as the database is encrypted with the master password.
3) Store hash of the master password and the salt for hash. Then compute and compare the hash of user input.

For the scheme 1, if an adversary can access the storage of the master password, the master password will be directly stolen, and all user data will be compromised. For the scheme 2, this authentication mechanism may be used as an interface for adversaries to perform a brute-force attack on the encrypted database. By comparison, for the scheme 3, if an adversary conducts a brute-force attack, the adversary may only find a collision of the master password. As a consequence, even if the adversary can pass the authentication mechanism, the database may not be decrypted. Therefore, PassSafe uses the scheme 3, and the hash function is HMAC-SHA-256.

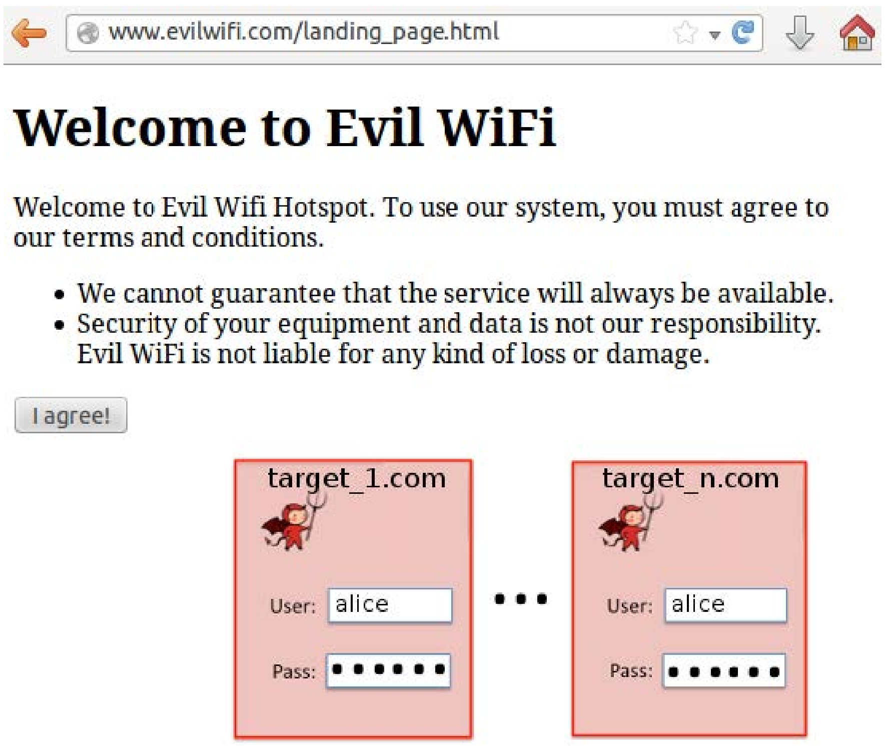
3.4 Autofill Function

Another core function is autofill which can significantly improve the convenience when using the password manager. To implement the autofill function, there are two problems should be tackled. The first one is how to identify the currently running app or website, and the second one how to input the relevant username and password to the specific fields. Five potential schemes will be discussed below.

Firstly, the crudest way is to use the clipboard, which is very easy to implement, while it requires much user interaction. When signing in, the user has to manually copy username and password and paste them into the relevant fields. The advantage of this scheme is that it does not need request any extra Android permission. However, it results a vulnerability is that any other applications on the device can access the clipboard without any permission. [29] indicates two categories of attack on Android clipboard, i.e., manipulation and stealing. If there are malicious applications on users' device, they can easily monitor the clipboard silently and steal users' credentials when they are copied to the clipboard. Therefore, any password manager should not use the Android clipboard to store or transfer users' credentials.

Secondly, the most widely used scheme is to implement an extra or built-in browser to cooperate the password manager. As the most of the traditional password managers on desktop computers are browser extensions or integrated into the browser, autofill technology in the browser is relatively mature. The password manager in a browser can directly analyse the URL of the website and obtain the domain to determine which pair of username and password should be filled. Then it parses the webpage and fills the login form. On the Android platform, developers can develop APIs for the browser and password manager to communicate with each other. Consequently, this scheme may also do not require other permission, and it could be the most automatic autofill which needs very little user interaction. However, it has three remarkable drawbacks. Firstly, it is heavyweight. The developers have to develop a complete browser, and the users have to give up their accustomed browsers. Secondly, it is constrained. On mobile devices, users often login on other third-party application rather than only websites in the browser. Finally, it is vulnerable to a phishing attack. For example, [30] shows a simple attack model (Figure 3.4). Over half of tested browser-based password managers are vulnerable to it, and user's passwords can be automatically exacted without any user interaction.

The third scheme is to implement the *InputMethodService* in Android to create an


www.evilwifi.com/landing_page.html

Welcome to Evil WiFi

Welcome to Evil Wifi Hotspot. To use our system, you must agree to our terms and conditions.

- We cannot guarantee that the service will always be available.
- Security of your equipment and data is not our responsibility. Evil WiFi is not liable for any kind of loss or damage.

Figure 3.4: A sample landing page of a rogueWiFi hotspot containing invisible iFrames to the target sites. Note that the iFrames are actually invisible to the user and shown here only for clarity.

input method editor (IME), which enables users to enter text. In the other word, it will be a soft keyboard. In addition to traditional functionalities of a keyboard, it also provides some keys for users to choose an account saved in the password manager and then help users to enter the username and password. In comparison with the clipboard, it is a service directly provided by the password manager. Hence other applications cannot monitor the data in it. Therefore, it is more secure than the clipboard.

The fourth scheme is to develop an Accessibility Service. An accessibility service is an API provided by Android that assists users with disabilities, or who may temporarily be unable to interact with a device fully. With given permissions by users, an accessibility service is able to monitor interface on the device and perform some actions on behalf of users. Therefore, to implement autofill function, the password manager application can develop an accessibility service which monitors widgets on the interface, and if a *password EditText* widget is detected, it may enter a password into that field depending on other information on the interface. Autofill based on this scheme could be highly automatic, while the decrease of user interaction will also reduce the resistance to a phishing attack. What is worse is that the accessibility service has to use Paste Action to enter text that means the clipboard is used to transfer confidential data. As we discussed above, the clipboard is dangerous.

For the last scheme, Android O (version 8.0) provides an Autofill Framework[2] which can improve user experience by autofilling forms in their devices. In this framework, the autofill service can manage autofill data and fills other apps' fields, and the password manager application can play the role of it. However, Android O is a Developer Preview edition by now, and its final public release is planned for Q3 2017. If this project is developed on Android O, most of the users' devices will not be able to run it immediately.

In conclusion, scheme three, a soft keyboard, can have better usability and security. Hence this scheme is applied in PassSafe. Many existing password managers also use this scheme, such as 1Password [5] and Puff [24], but most of them have a significant drawback is that every time users have to find and select the target password note from possibly a large number of notes, which is time-consuming and inconvenient. Therefore, this project tries to solve this problem that users can receive a precise suggestion when they need to select a note. Implementation details will be introduced in section 4.8.

[2]https://developer.android.com/preview/features/autofill.html

Chapter 4

Implementation

This chapter describes the implementation details of PassSafe, the Android password manager application developed in this project. It begins with presenting the general work flow of the app, and then it shows the interfaces and internal implementation of 7 primary components, i.e. Master Password Setting Activity, Authentication Activity, Main Activity, View Activity, Edit Activity, Password Generator Activity and Autofill Soft Keyboard Service.

4.1 System Overview

When the master password has not been set (e.g. first opening of the app), the application will guide the user to set a master password, and the database will be initialised. After the app is initialised, whenever the user opens this app, it will verify user's master password. If the user can successfully log in, the main page will be presented, which can dynamically show a list of user's account information. And then the user can add a new account note or view details of an account note. Meanwhile, the user can edit, delete it or set it as favourite. When the user is adding or editing an account note, a password generator is provided to generate strong, secure and customised passwords. A flow chart is shown in Figure 4.1.

4.2 Master Password Setting Activity

Whenever the app is opened, it will check whether the master password has been set. If not, for example, when the app is just installed, and the user opens the app for the first time, it will guide the user to set a strong master password. The interface can be

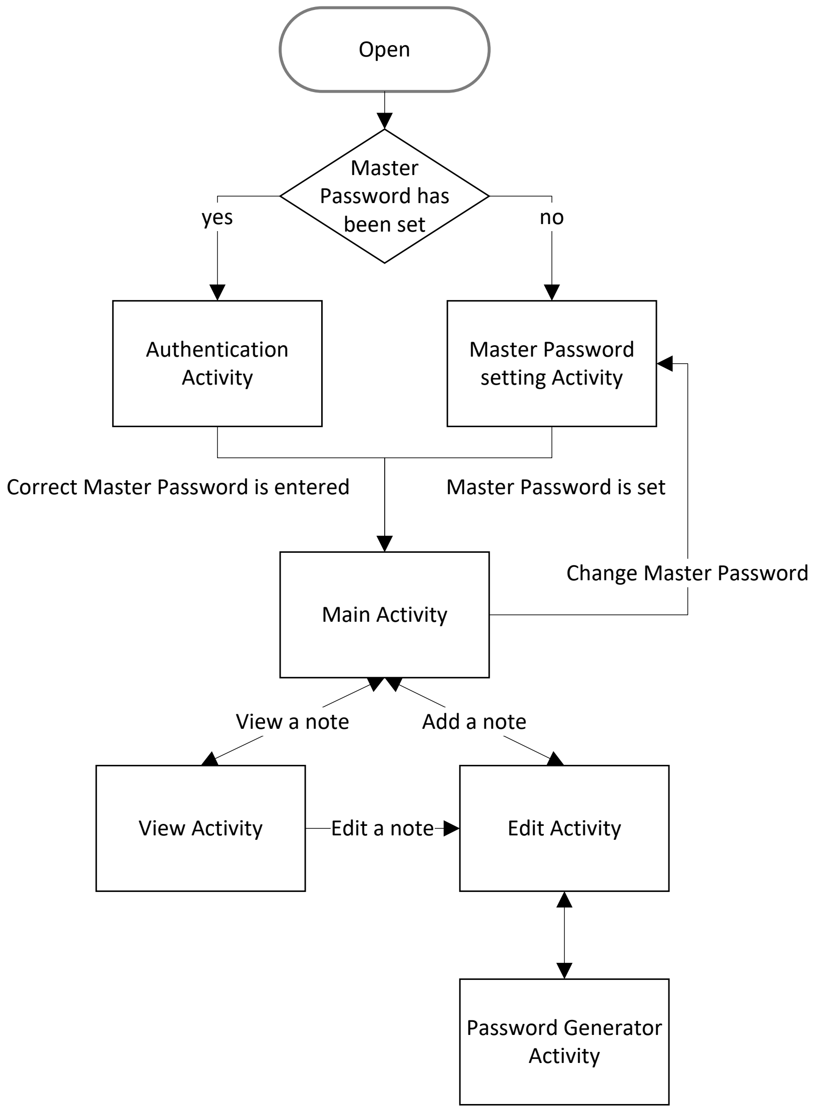
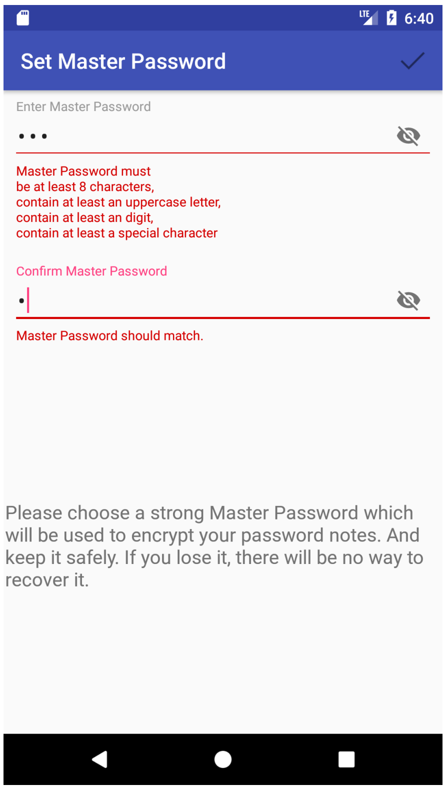


Figure 4.1: Overview flow chart

seen in Figure 4.2.



Master Password must
be at least 8 characters,
contain at least an uppercase letter,
contain at least an digit,
contain at least a special character

Master Password should match.

Please choose a strong Master Password which will be used to encrypt your password notes. And keep it safely. If you lose it, there will be no way to recover it.

Figure 4.2: Interface of Master Password Setting Activity

The interface is mainly composed by two *TextInputLayouts*. Each of them wraps an *EditText*. It can help to show a floating hint when the hint is hidden by the inputting text from users, and to deploy a visibility toggle button to show password between plain-text and disguised. Also, it can display an error message. Because the master password will be used to encrypt the database and authenticate the user, its strength is critical for the security of the app, but many users are used to choosing a password with low entropy which may be easy to remember. Therefore, the app helps the user to set a sophisticated enough master password. The error message is used to show the requirements of master password dynamically. Whenever the input text is changed, it will be parsed to check the remaining requirements (Figure 4.3), and the requirements are set as the error message to be dynamically shown under the first input box. Similarly, the error message under the second input box will be shown when the inputs do not match. Only when there is no error about inputs, the user is able to save the master password.

As we discussed in chapter 3.3, the master password should be hashed, and the result of hash and the salt for hashing should be stored. For the hash function, HMAC-

```java
StringBuilder errorMsg = new StringBuilder("Master Password must\n");
if (charSequence.length() < 8) {     errorMsg.append("be at least 8 characters,\n");}
for (int i = 0; i < charSequence.length(); i++) {
    hasUpper = hasUpper || Character.isUpperCase(charSequence.charAt(i));
    hasLower = hasLower || Character.isLowerCase(charSequence.charAt(i));
    hasDigit = hasDigit || Character.isDigit(charSequence.charAt(i));
    hasSpecial = hasSpecial || !Character.isLetterOrDigit(charSequence.charAt(i));
}
if (!hasUpper) { errorMsg.append("contain at least an uppercase letter,\n");}
if (!hasLower) { errorMsg.append("contain at least a lowercase letter,\n");}
if (!hasDigit) { errorMsg.append("contain at least an digit,\n");}
if (!hasSpecial) { errorMsg.append("contain at least a special character,\n");}
```

Figure 4.3: Code for generating the error message

SHA-256 is used in PassSafe. Android only supports HMAC-SHA-1, while SHA1 is not secure. Therefore, an external library, *Spongy Castle*[1], is used to implement HMAC-SHA-256, which is a repackage version of *Bouncy Castle*[2] for Android. Bouncy Castle is a widely used open-sourced collection of APIs for cryptography. Android also includes a customised version of Bouncy Castle. The implementation of the hash function can be seen as Figure 4.4. The derived hash length is defined as 256 bits. Because a time-consuming hash process can significantly increase resistance to a brute-force attack, and this process will barely happen, it is reasonable to set a relatively high iteration count. With experiments on hard devices, when the iteration count is 104, the hash process will cost from 1 to 10 seconds, which is acceptable for users when initialising or changing the master password. In addition, to avoid main UI thread blocking and ANR (Application Not Response) the hash process runs in another thread.

```java
SecureRandom random = new SecureRandom();
salt = new byte[saltLength];
random.nextBytes(salt);
PKCS5S2ParametersGenerator generator =
        new PKCS5S2ParametersGenerator(new SHA256Digest());
generator.init(PBEParametersGenerator.
        PKCS5PasswordToUTF8Bytes(masterPass.toCharArray()), salt, iterationCount);
KeyParameter key = (KeyParameter) generator.generateDerivedMacParameters(keyLength);
keyByte = key.getKey();
```

Figure 4.4: Code for the Hash function

Considering to store the hash and the salt, Android provides several options for data

[1]Spongy Castle: https://rtyley.github.io/spongycastle/
[2]Bouncy Castle: https://www.bouncycastle.org/

storage, i.e. Shared Preference, Internal Storage, External Storage, SQLite Databases and Network connection.

Shared Preference provides a lightweight framework to store and key-value pairs of primitive data types, which is suitable for storing salt and hash result, while Internal or External Storage and SQLite Databases are more suitable for storing a large amount of data, as which have to create and read more complicated files. Network connection could have better security performance and flexibility, but this app is entirely offline now. In future work, a server could be built up to store these data. Therefore, Master Password Setting Activity writes the hash and the salt on shared preference, and Authentication Activity retrieves them from shared preference.

Main Activity provides an interface to change master password. Consequently, this activity is also responsible for resetting master password, and the interface should be slightly different between initialising and resetting. To indicate this activity which function should work, when starting the activity, extra information is put in intent to pass the message between activities (Figure 4.5).

```
Intent intent = new Intent();
intent.setClass(MainActivity.this, MasterPasswordSettingActivity.class);
intent.putExtra("isInitialising", false);
startActivity(intent);
```

Figure 4.5: Code for transferring message

4.3 Authentication Activity

If the master password has been set, the app will always begin with authentication activity to authenticate the user. The interface is shown as Figure 4.6.

The interface is very concise that there is only a *TextInputLayout* and an Authenticate button. User input will be hashed by the same method with the Master Password Setting Activity and with the stored salt. The result of hash will be compared with the stored hash of the master password to authenticate the user. If a wrong password is entered, an alert box will pop up to warn the user. To prevent brute-force attack on authentication mechanism, five consecutive wrong password inputs will cause the destruction of all user data (i.e. database and shared preference file). After a wrong password is input, an error message will be shown under the input box to warn the number of chances left.

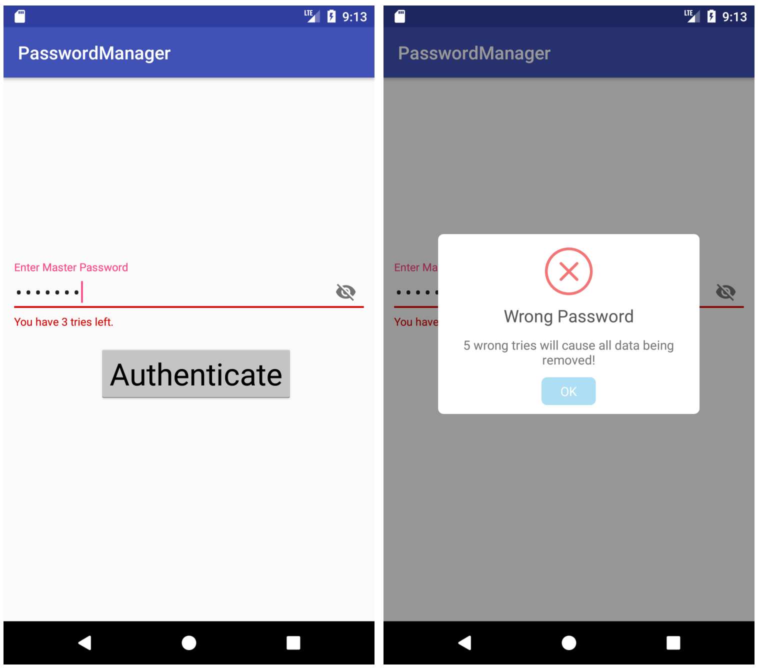


Figure 4.6: Interface of Authentication Activity

4.4 Main Activity

If the user is authenticated, the main activity will start as Figure 4.7. The interface shows a list of password notes saved by the user. The list could be very long when the user stores a large number of password notes, but there is only very limited space on the interface. Therefore, it is implemented by a *RecyclerView*, which is a more advanced and flexible version of *ListView*. This widget is suitable for displaying large data sets that can be scrolled very efficiently. In the *onCreate()* method of activity, the *RecyclerView* is attached with an adapter which manages the data to be shown, creates views for items of data and attaches reaction for clicking on an item in the list (Figure 4.8).

To improve user experience, every password note is shown with its note name and username. They are sorted by note names in ascending order and grouped by the first letter of note name. The first letter is defined as the group name showed above each group on the colourful stick header. Favourite notes are placed at the top of the list as “☆” group. In addition, names begin with not a letter are placed together (as “#” group) to make the list not too messy. According to *Flexibility and efficiency of use* from Nielsen’s 10 heuristics[31], on the right side of the interface, there is a navigation side bar where if a letter is touched, the list will be scrolled to that group for the user

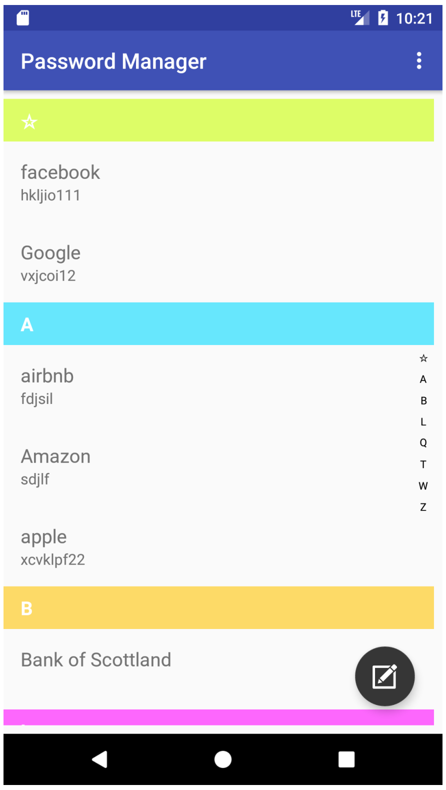


Figure 4.7: Interface of Main Activity

```
rcyNotes = (RecyclerView) findViewById(R.id.rcy_notes);
rcyNotes.setLayoutManager(new LinearLayoutManager(this));
PasswordNoteAdapter.NoteClickListener noteClickListener =
        new PasswordNoteAdapter.NoteClickListener() {
    @Override
    public void onNoteClick(int position) {
        PasswordNote note = noteAdapter.getNote(position);
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ViewActivity.class);
        intent.putExtra("id", note.getId());
        startActivityForResult(intent, 0);
    }
};
noteAdapter = new PasswordNoteAdapter(noteClickListener);
rcyNotes.setAdapter(noteAdapter);
```

Figure 4.8: Code for setting the RecyclerView in Main Acitivity

to find a note quickly.

A floating action button is located at the right bottom for adding a new password note. When the user is scrolling the list down, the floating button will hide automatically in case of affecting user to click a note. The button will appear again when the list is scrolled up. More settings are collected on the right top, such as changing the master password and enabling the soft keyboard.

Content and order of the list may change with editing, deleting or adding a password note. However, it is resource consuming to update the list frequently regardless of whether there are changes, or inconvenient for users to update it manually. Therefore, we use *startActivityForResult* (Figure 4.8) and *onActivityResult* (Figure 4.9) methods in Main Activity and setResult method in other methods to tackle that. When View Activity or Edit Activity started by Main Activity are finished, they will inform Main Activity whether there are changes and the list should be updated.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (resultCode) {
        case RESULT_OK:
            updateNotes();
            break;
        default:
    }
}
```

Figure 4.9: Code of onActivityResult

4.5 View Activity

When the user clicks a note in Main Activity, View Activity will be activated to show the details of this password note (Figure 4.11). In addition to the user-entered website, login username, password and note, there are system-generated created, modified and accessed time to help users to have full control of their confidence.

Two buttons are located on the toolbar to edit or delete this note. The Star Button is for the user to set this note favourite, which will be at the top of the list in Main Activity. The Open Button can directly open the website in a browser selected by the user. To open the website in a password note, we need to parse the URL from the user and use the result to start an activity (Figure 4.10).

The format of a URL is very strict:

```
    Intent intent = new Intent();
    intent.setAction(Intent.ACTION_VIEW);
    Uri uri = Uri.parse(URLInputByUser);
    intent.setData(uri);
    startActivity(intent);
```

Figure 4.10: Code for open the website in a password note

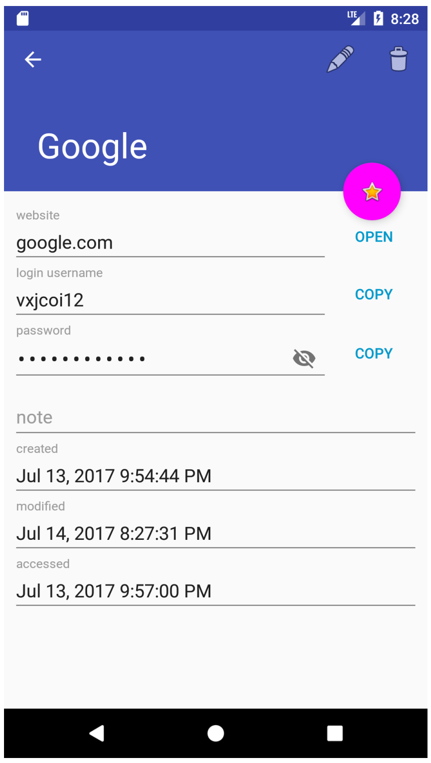


Figure 4.11: Interface of View Activity

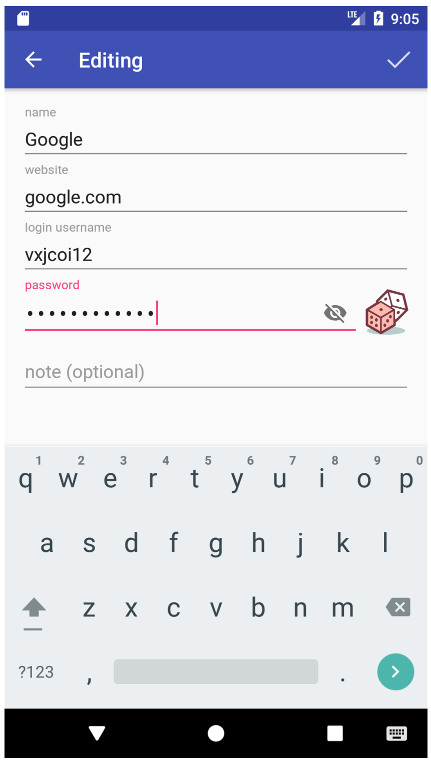


Figure 4.12: Interface of Edit Activity

scheme:[//[user[:password]]host[:port]][/path][?query][#fragment]
But the URL input by users are invalid at most of time. “https://www.google.com” for example, a user may input “www.google.com”, “google.com” or just “google”. Therefore, it need to be cleaned up and reformatted. Android provides a useful API, *android.webkit.URLUtil*, in which a method *guessUrl(String inUrl)*[3] can do this well. The examples above can all be parsed correctly.

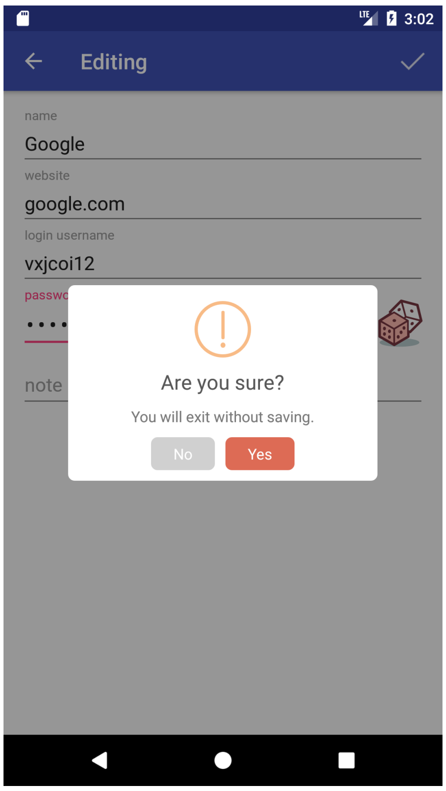
As discussed above, the clipboard in Android is vulnerable, so PassSafe tries to avoid users using it to transfer their confidence. The two Copy Buttons will copy the username and password to the soft keyboard, and the clipboard will be not involved in.

[3]https://developer.android.com/reference/android/webkit/URLUtil.html

4.6 Edit Activity

The interface (Figure 4.12) of Edit Activity is composed by 5 TextInputLayouts for users to input information of a password note. Adding a new note and editing an existing note both use Edit Activity. Therefore, similar to Master Password Setting Activity, an extra flag is added into the intent to indicate the valid function. The title of this activity will vary depending on the valid function. When adding a new note, input information will be saved as a new row inserted into the database. When editing an existing note, stored information will be loaded into the input boxes for the user to edit and finally the existing raw in the database will be updated. Next to the password input box, there is a dice button to open Password Generator Activity for the user to generate a customised secure password.

A warning box (Figure 4.13) is built up following *Error Prevention* in Neilson's 10 Heuristics. When a user is trying to exit the Edit Activity directly, the warning box will pop up for the user to double confirm, which can effectively prevent user unexpectedly quit without saving their inputs.



Are you sure?

You will exit without saving.

Figure 4.13: Waring box when exit Edit Activity

4.7 Password Generator Activity

The random number generator (RNG) is a crucial part of the password generator in PassSafe, and the performance of it can considerably affect the security. [32] defines the requirement of RNG for cryptographic applications:

> (R1) The random numbers should have good statistical properties.
>
> (R2) The knowledge of subsequences of random numbers shall not allow one to practically compute predecessors or successors or to guess these numbers with non-negligibly larger probability than without knowledge of these subsequences.

Two RNGs are provided in Android, java.util.Random [33] and java.security.SecureRandom [34]. Both of them can satisfy Requirement 1, while java.util.Random does not satisfy Requirement 2. It uses a linear congruential formula to modify a seed to generate random numbers. However, the output of the linear congruential generator is predictive [35]. In addition, the length of the output sequence is 48 bit, which is too short to be very vulnerable to brute-force attack. In comparison, java.security.SecureRandom can generate 128-bit cryptographically strong random numbers by using /dev/random or /dev/urandom created from the kernel entropy pool. The kernel entropy pool is collected from a hardware random number generator which can generate true random numbers. Consequently, java.security.SecureRandom is applied in PassSafe as the RNG.

The hash function is implemented as same as Figure 4.4, but if the iteration number is more than 10^4, there will be a significant delay. Consequently, the iteration count is set as 10^3. For the flexibility, users can choose whether to include uppercase letters, lowercase letters, numbers or symbols. In addition, there are some similar characters (e.g. i, I, l, L, 1). If a user has to input the password on another device manually, it may be difficult for the user to distinguish the similar characters and input the password correctly. Therefore, users can also choose whether to exclude the similar characters. Finally, a dictionary is composed of the characters chosen by users. The password generator will randomly select characters in the dictionary with the result of the hash function:

$$character = dictionary.charAt(randomNumber \bmod dictionary.length)$$

Those characters will be concatenated as the generated password. A key derived by HMAC-SHA-256 is 64 bytes. The range of a byte is 0-255. If we use one byte to select a character, the possibility of every character to be selected will be significantly different. For example, if uppercase and lowercase letters and numbers are chosen, there will be 62 characters in the dictionary, and the possibility of each of the first 8 characters to be selected will be 25% bigger than each of others'. Hence, a two-byte number is used to select a character, which can significantly decrease the unbalance. A key derived by HMAC-SHA-256 can be used to select at most 32 characters. However, if a password whose length is longer than 32 is required, more keys will be generated to select characters.The code for the encode function can be seen in Figure 4.14.

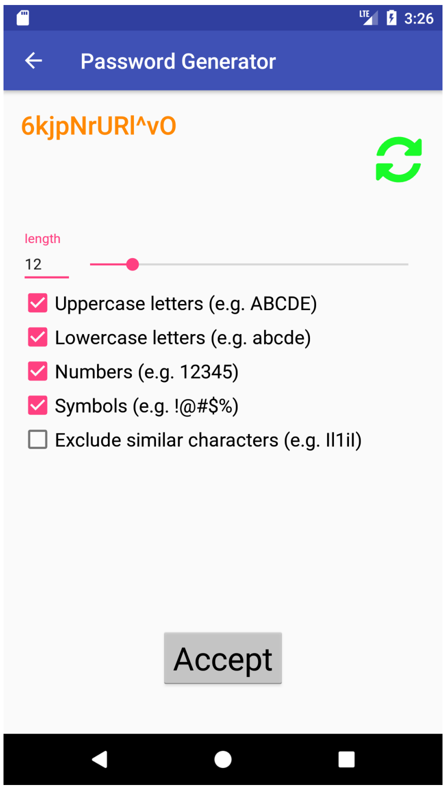
```
int i = 0;
while(i < length){
    byte[] keyByte = hashUtil.deriveHash();
    int j = 0;
    while (i < length & j < keyByte.length) {
        byte[] temp = new byte[2];
        temp[0] = keyByte[j];
        temp[1] = keyByte[j+1];
        ByteBuffer wrapped = ByteBuffer.wrap(temp);
        short rand = wrapped.getShort();
        pw.append(dict.charAt((rand & 0xFFFF) % dict.length()));
        j += 2;
        i += 1;
    }
}
```

Figure 4.14: Code for the encode function

The interface of this activity is shown as Figure 4.15. The generated password will be displayed at the top of the screen. The green renew button is for regenerating a new password if settings of the password are changed, or the user is not satisfied with current password. The user can directly enter the length of the password into the box, or touch the thumb on the progress bar and drag left or right to change the length. In addition, there are five check boxes for customising the components of the password. Finally, the user can click accept button, and the generated password will be filled into the password box of Edit Activity.

4.8 Autofill Soft Keyboard Service

The Autofill Soft Keyboard Service is extended from *InputMethodService* which provides a basic framework for user interface and internal functions of the keyboard. The



6kjpNrURl^vO

length
12

☑ Uppercase letters (e.g. ABCDE)
☑ Lowercase letters (e.g. abcde)
☑ Numbers (e.g. 12345)
☑ Symbols (e.g. !@#$%)
☐ Exclude similar characters (e.g. Il1il)

Figure 4.15: Interface of Password Generator Activity

interface is designed similar with system default keyboard to make user adapt to it easily, which can be seen as Figure 4.16.

When users need to enter a pair of username and password, they just click the Safe Button (the first button in the first row). After verifying the master password, if the current application has been autofilled before, a suggestion will pop up for users to select as Figure 4.17. Otherwise, users select a desired password note manually. Then users can click the User Button and Key Button (the second and third buttons) to enter the selected username and password.

To implement the quick suggestion function, currently running foreground application should be identified. *android.app.ActivityManager* API provides a method *getRunningTasks* which can return a list of tasks that are currently running, with the most recent being first. When users click the Safe Button, this method will be called, and the package name of the foreground application will be obtained. If there is a saved note with the package name, it will pop up as the suggestion. Otherwise, after users selecting a password note, the package name will be saved in the selected password note. However, the getRunningTasks method was deprecated in API level 21 (Android version 5.0) for privacy problems. If the users' devices use higher version of Android,

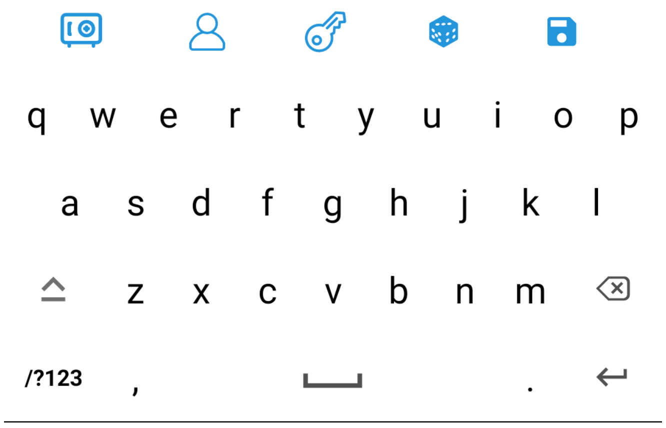


Figure 4.16: Interface of Soft Keyboard

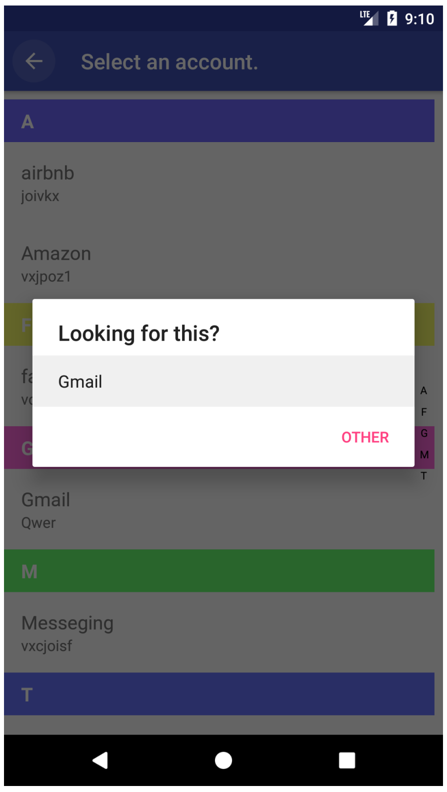


Figure 4.17: Suggestion when user is selecting an password note for autofill

another strategy should be employed. A new API *UsageStatsManager* was added in API level 21, which provides access to device usage history and statistics, where the *queryUsageStats* method can return a list of application usage stats. The stat contains the last time an application was used. Therefore, the application with the shortest time between the last time used and the current time is the current foreground application. Part of code can be seen in Figure 4.18.

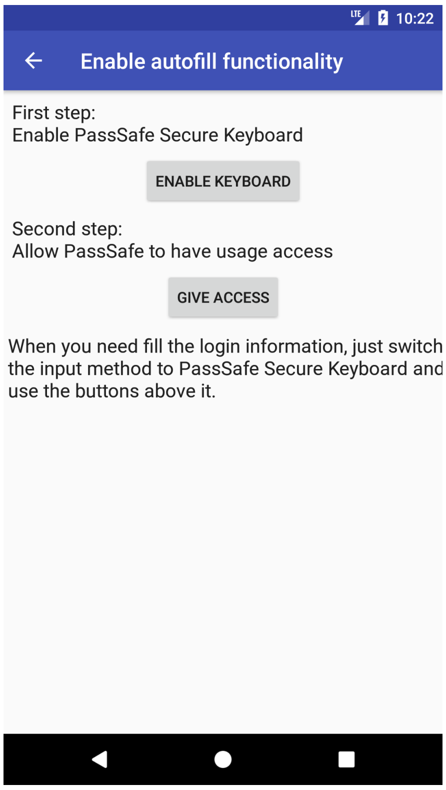
```
class RecentUseComparator implements Comparator<UsageStats> {
    public int compare(UsageStats lhs, UsageStats rhs) {
        return (lhs.getLastTimeUsed() > rhs.getLastTimeUsed())
                ? -1 : (lhs.getLastTimeUsed() == rhs.getLastTimeUsed()) ? 0 : 1;
    }
}
RecentUseComparator mRecentComp = new RecentUseComparator();
long ts = System.currentTimeMillis();
UsageStatsManager mUsageStatsManager = (UsageStatsManager) context
        .getSystemService(Context.USAGE_STATS_SERVICE);
List<UsageStats> usageStats = mUsageStatsManager
        .queryUsageStats(UsageStatsManager.INTERVAL_BEST, ts - 1000 * 3600 * 12, ts);
Collections.sort(usageStats, mRecentComp);
return usageStats.get(0).getPackageName();
```

Figure 4.18: Code for getting package name of foreground running application after API level 21

However, there is a drawback that when users are trying to log in a website in the browser, the password manager application can only obtain the package name of the browser application. It cannot know what website the user is visiting. Although some other implementations, such as [8], tried to identify the currently visited website by reading the top item from the browser's history with Android's READ_HISTORY_BOOKMARKS permission, this permission has been removed from API level 23 (Android version 6.0). Furthermore, there is no alternative way to implement the same function now. Consequently, the suggestion may provide a list of accounts that have been signed in on the browser.

The rest two buttons in the first row of the soft keyboard provide access to the password generator and a quick save feature. When users are registering an account, they can click the User Button to save the entered username, click the Dice Button to open the Password Generator and generate a random password, and finally click the Save Button to open the Edit Activity for quickly adding a password note. The name of note, username and password can be automatically filled into the Edit Activity. The name of note is derived from package name of currently running application. The package name is normally a concatenation of the reversed domain of a company and the application name, e.g. “com.android.chrome”. Therefore, the package name is split by “.” and the last part can be set as the name of the note.

To use the autofill function, users should switch the input method to the soft keyboard of PassSafe, and to enable the suggestion for autofill, users should give PassSafe the permission to access *UsageStatsManager*. To help users complete those settings conveniently, there is an Enable Autofill Activity (Figure 4.19) entered from the Main Activity. The two button can directly open the related setting pages in the operating system.



First step:
Enable PassSafe Secure Keyboard

Second step:
Allow PassSafe to have usage access

When you need fill the login information, just switch the input method to PassSafe Secure Keyboard and use the buttons above it.

Figure 4.19: Interface of Enable Autofill Activity

Chapter 5

Evaluation

This chapter demonstrates the two evaluations for PassSafe. Section 5.1 presents the Think-aloud usability evaluation with five tasks given to the participants to test the core functionalities of PassSafe and proposes the improvement schemes for feedback from the test. Section 5.2 illustrates the feature-based comparison of PassSafe with other six Android password manager apps.

5.1 Usability Evaluation

To test the usability of PassSafe and to make sure it can work well under the real-world circumstance, Think-aloud Protocol is used in this project. It is a very commonly employed in usability evaluation, whose basic idea is to have a participant to use the system and speak aloud while they do so. Advantages of this methodology are that researcher can get a sense of what the user is trying to do and why they click on some things and testing with five users may find the majority of major issues.

Our experiments are in accordance with ethics policy of the University of Edinburgh and all participants signed the consent form given in appendix A. 5 participants finally took part in this research, three of them are new to password managers, and two of them have used other password manager apps before. An android phone (Meizu PRO6, Android 6.0) was used in the research, which was installed with PassSafe and a mock-up app made by us. The mock-up app only has a sign-in and a register interfaces, hence participants will not have concerns and risks for exposing their credentials. We began with giving participants a brief introduction to this application and an explanation of the study and told they should be honest about their impressions of the application because we were testing the application, not their performance. Then we gave them a think-aloud pre-training to make them adapt to think-aloud protocol. Finally, five tasks were given to participants for them to complete to test the core functionalities of the app. The script can be seen in Appendix B.

1) Open PassSafe at the first time and set the master password to initialise it. If it is successfully initialised, close it and try to open and log in it again.

2) Add a new password note with given information and use the password generator to generate a random password.

3) Open, view and edit a password note, and finally delete it.

4) Enable the autofill function, and use the autofill to login in the mock-up app.

5) Register an account in the mock-up app and use the quick save feature to save the account information.

Overall, all participants could complete the first three tasks successfully. There are some minor problems. In task 1, two participants complained the requirements of the master password are too complicated for them, while we believe the strength of master password is a key element for the security of password manager as it will be used to authenticate users, encrypt database and even generate passwords in PassSafe. If we do not require users to set a strong master password, many of them may choose a weak password with low entropy, which could significantly decrease the security of this system. In task 2, three participants felt confused at the beginning, because when users have not added any note, there will be a large blank in the Main Activity as left figure in Figure 5.1. Therefore, we improved the Main Activity that it could show some instructions when there is no note in the list as the right figure in Figure 5.1, and participants provided these instruction would be very meaningful for them. In task 3, two participants thought the name attribute was ambiguous for them, because they did not know it was the name of this note, name of the site or name of the account. Consequently, we changed the name attribute to name of this note to make it more clear for users.

The last two tasks were more challenging. The participants with experience of using other password managers could accomplish them successfully, while other participants spent more time, and one of them enabled the autofill but failed for the rest of the tasks. The primary reason is that they had not ever used soft keyboard to autofill before. Hence they were unfamiliar with this mechanism. In addition, the autofill and

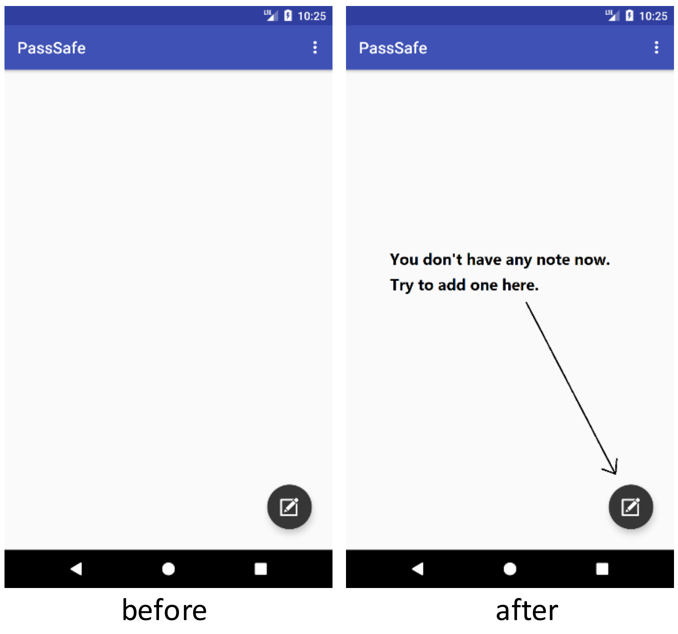


Figure 5.1: Main Activity when there is no note.

quick save functions are not entirely automatic and require some specific user interaction. To tackle this problem, in the future, we would develop a welcome page to provide a general guide and core ideas of the app, which will show when user open the app at the first time. And we may also provide a help document for users. Furthermore, increasing automaticity and reducing user interaction are another important future work.

5.2 Feature-based comparison with other password managers

Finally, PassSafe is compared with six paid and free Android password manager apps on some primary features. We firstly listed whether it is open-source. Then we researched their schemes for password generator, database encryption, autofill and synchronisation which are core components of a password manager. Because users may have various devices (e.g. desktop or mobile, Android or IOS), multiplatform support is important for the accessibility and availability of a password manager. We also checked whether they involve the clipboard which will decrease the security significantly. Finally, we explored distinctive features they have. The results are shown in

Table 5.1.

Table 5.1: Comparison table of PassSafe with other 6 Android password manager apps

Password Manager	Open-source	Password generator	Database Encryption	Autofill	Synchronisation	Multi-platform support	Clipboard involvement	Distinctive features
1Password[5]	N	Random	AES-256	Accessibility Service + Soft Keyboard	Server	Y	Y	
LastPass[6]	N	Random	AES-256	Accessibility Service + Browser	Server	Y	Y	LastPass Authenticator
Dashlane[7]	N	Random	AES-256	Accessibility Service + Browser	Server	Y	Y	Password changer
KeePassDroid[23]	Y	Random, Scheme A	AES-256	N	Manually export and import database	Y	Y	
PadLock[25]	Y	Random, Scheme A	AES-256	N	Cloud drive	Y	Y	
Puff[24]	Y	Random, Scheme A	Blowfish	Soft Keyboard	Can only export database	N	Y	
PassSafe	Y	Random, Scheme C	AES-256	Soft Keyboard + Suggestion	N	N	N	Quick save

All three paid password managers are closed-source, while the other four free ones are open-source. For the password generator scheme, all of the apps generate random passwords. Since the first three are closed-source, and it is not specified in their security white papers, we cannot know their schemes in details. The other three open-source app use scheme A which is described and proved not secure in section 3.1, while only PassSafe uses scheme C which may have better security performance. Most of the apps including PassSafe encrypt their databases with AES-256, except Puff with Blowfish.

Regarding Autofill, all paid apps have two alternative schemes, while two free apps do not have autofill function and the other two use soft keyboard. There is one notable point that among apps with autofill by soft keyboard only PassSafe can give a suggestion account to fill, while others have to choose an account manually.

Paid apps have better performance on synchronisation and multi-platform support. They store users' data on their servers and users' can retrieve their data in any devices through their accounts, but this scheme may be too expensive for free apps. A primary drawback of PassSafe is that it does not support synchronisation by now, which is the most imperative future work for us.

A security issue, particularly for Android, is researched in the comparison. As we discussed in section 3.4, the clipboard on Android is accessible by any apps on the device without requiring any permissions, hence a malware on the device can monitor the clipboard silently in the background. If the password manager copies passwords to the clipboard, users' credentials could be easily stolen by malware. Therefore, a password manager should not send data to the clipboard and should try to avoid users copying their confidence to the clipboard. However, all apps only except PassSafe provide buttons or other means for users to copy usernames and passwords to the clipboard. By comparison, when users click the copy buttons in the PassSafe, it still keeps the data in its own process which is isolated with other apps, and users can use the soft keyboard to output the data.

For distinctive features, LastPass has a unique Authenticator with two-factor authentication, and it supports 6-digit generated passcodes, SMS codes and automated push notifications for authentication. The Password Charger in DashLane can help users automatically change their passwords on hundreds of popular sites. PassSafe also provides a distinctive feature, Quick save, which has been introduced in section 4.8, while other apps cannot help when users are registering an account. After they finished the registration, they have to manually input the account information in the password manager again, which is inconvenient.

Chapter 6

Conclusion

Many users are struggling with managing their large amount of text passwords. Hence they may reuse their passwords or/and use weak passwords to reduce their burdens. However, these strategies will make users' password security critically vulnerable. Password managers are introduced to tackle this problem by generating unique passwords and retrieving passwords for users. There are many password manager apps on Android by now, which can provide good accessibility and mobility, but most of them have some serious security flaws. In addition, we believe more features could be applied to a password manager to make it more convenient. Therefore, we proposed the development of an Android password manager application.

We reviewed the development of password managers in both literature and practical software. Furthermore, we analysed multiple schemes for the design of primary components in our application. Finally, an open-source Android password manager application, PassSafe, was developed by us. This application can generate random passwords with an improved scheme, store users' passwords with strong encryption and automatically fill usernames and passwords into third-party apps. We believe PassSafe can provide a user-friendly interface with a think-aloud usability evaluation and competitive functionality and security with a comparison with other android password managers.

6.1 Future work

In the future, it is imperative to develop the synchronisation function in PassSafe. For the first step, PassSafe should be able to export and import the database file, and users could synchronise their database files, for example, by using a cloud drive. This method could be easy to deploy and require a very low cost. For the further future, we could deploy a server to store users' database, which could release users' burden for synchronisation and increase the security. When the app is entirely offline, if the device is obtained by an adversary, the adversary may bypass the authentication mechanism and directly exact and attack the database. By comparison, if the database is stored in the server, the loss of device may not result in the loss of database.

Another work is to improve the automaticity and phishing resistance of autofill function. To sign in on a site, users have to choose an account and click two buttons to fill the username and password respectively. We may simplify this process by enabling PassSafe to directly fill the username after users selecting an account, and automatically jump to the password field and fill the password. To protect users from phishing attacks, when users are choosing an account to fill a third-party app, we could show the company information and application name exacted from the app package name which can uniquely identify the app. However, when users are trying to sign in on a website in a browser, it is unable to exact the URL of the site unless the browser is built in PassSafe. Therefore, we may require users to copy the URL to PassSafe, and it notifies users whether this domain matches the website recorded in the password note.

Bibliography

[1] D. Florencio and C. Herley, “A large-scale study of web password habits,” pp. 657–666, 2007.

[2] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, “The tangled web of password reuse.” in *NDSS*, vol. 14, 2014, pp. 23–26.

[3] A. Karole, N. Saxena, and N. Christin, “A comparative usability evaluation of traditional password managers.” in *ICISC*. Springer, 2010, pp. 233–251.

[4] “Mobile/tablet operating system market share 2017,” https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1, 2017, [Online; accessed 1-August-2017].

[5] “1password,” https://1password.com/, 2017, [Online; accessed 6-April-2017].

[6] “lastpass,” https://www.lastpass.com/, 2017, [Online; accessed 6-April-2017].

[7] “Dashlane,” https://www.dashlane.com/, 2017, [Online; accessed 6-April-2017].

[8] S. Fahl, M. Harbach, M. Oltrogge, T. Muders, and M. Smith, “Hey, you, get off of my clipboard,” pp. 144–161, 2013.

[9] Z. Li, W. He, D. Akhawe, and D. Song, “The emperor’s new password manager: Security analysis of web-based password managers.” in *USENIX Security Symposium*, 2014, pp. 465–479.

[10] P. Gasti and K. B. Rasmussen, “On the security of password manager database formats.” in *ESORICS*. Springer, 2012, pp. 770–787.

[11] C. Herley and P. Van Oorschot, “A research agenda acknowledging the persistence of passwords,” *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[12] J. Kelsey, B. Schneier, C. Hall, and D. Wagner, "Secure applications of low-entropy keys," in *International Workshop on Information Security.* Springer, 1997, pp. 121–134.

[13] A. H. Karp, "Site-specific passwords," *HP Labs Technical Report*, 2002.

[14] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions." in *USENIX Security Symposium.* Baltimore, MD, USA, 2005, pp. 17–32.

[15] J. A. Halderman, B. Waters, and E. W. Felten, "A convenient method for securely managing passwords," in *Proceedings of the 14th international conference on World Wide Web.* ACM, 2005, pp. 471–479.

[16] B. Stock and M. Johns, "Protecting users against xss-based password manager abuse," in *Proceedings of the 9th ACM symposium on Information, computer and communications security.* ACM, 2014, pp. 183–194.

[17] "Bug 360493 - cross-site forms + password manager = security failure," https://bugzilla.mozilla.org/show_bug.cgi?id=360493, 2006, [Online; accessed 1-August-2017].

[18] M. Blanchou and P. Youn, "Password managers: Exposing passwords everywhere," https://raw.githubusercontent.com/iSECPartners/publications/master/whitepapers/password_managers.pdf, 2013, [Online; accessed 1-August-2017].

[19] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot, "Tapas: design, implementation, and usability evaluation of a password manager," in *Proceedings of the 28th Annual Computer Security Applications Conference.* ACM, 2012, pp. 89–98.

[20] F. A. Maqbali and C. J. Mitchell, "Autopass: An automatic password generator," *arXiv preprint arXiv:1703.01959*, 2017.

[21] E. Stobert and R. Biddle, "A password manager that doesn't remember passwords," in *Proceedings of the 2014 workshop on New Security Paradigms Workshop.* ACM, 2014, pp. 39–52.

[22] M. D. Leonhard and V. Venkatakrishnan, "A comparative study of three random password generators," in *Electro/Information Technology, 2007 IEEE International Conference on.* IEEE, 2007, pp. 227–232.

[23] "Keepassdroid," http://www.keepassdroid.com/, 2017, [Online; accessed 13-June-2017].

[24] "Puff," https://github.com/PuffOpenSource/Puff-Android, 2017, [Online; accessed 14-June-2017].

[25] "Padlock," https://padlock.io/, 2017, [Online; accessed 14-June-2017].

[26] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs, "Security analysis of pseudo-random number generators with input:/dev/random is not robust," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 647–658.

[27] B. Kaliski, "Pkcs# 5: Password-based cryptography specification version 2.0," 2000.

[28] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," Cryptology ePrint Archive, Report 2017/190, Tech. Rep., 2017.

[29] X. Zhang and W. Du, "Attacks on android clipboard," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 72–91.

[30] D. Silver, S. Jana, D. Boneh, E. Y. Chen, and C. Jackson, "Password managers: Attacks and defenses." in *USENIX Security Symposium*, 2014, pp. 449–464.

[31] J. Nielsen and R. Molich, "Heuristic Evaluation of user interfaces," *CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, no. April, pp. 249–256, 1990.

[32] W. Schindler, "Random number generators for cryptographic applications," in *Cryptographic Engineering*. Springer, 2009, pp. 5–23.

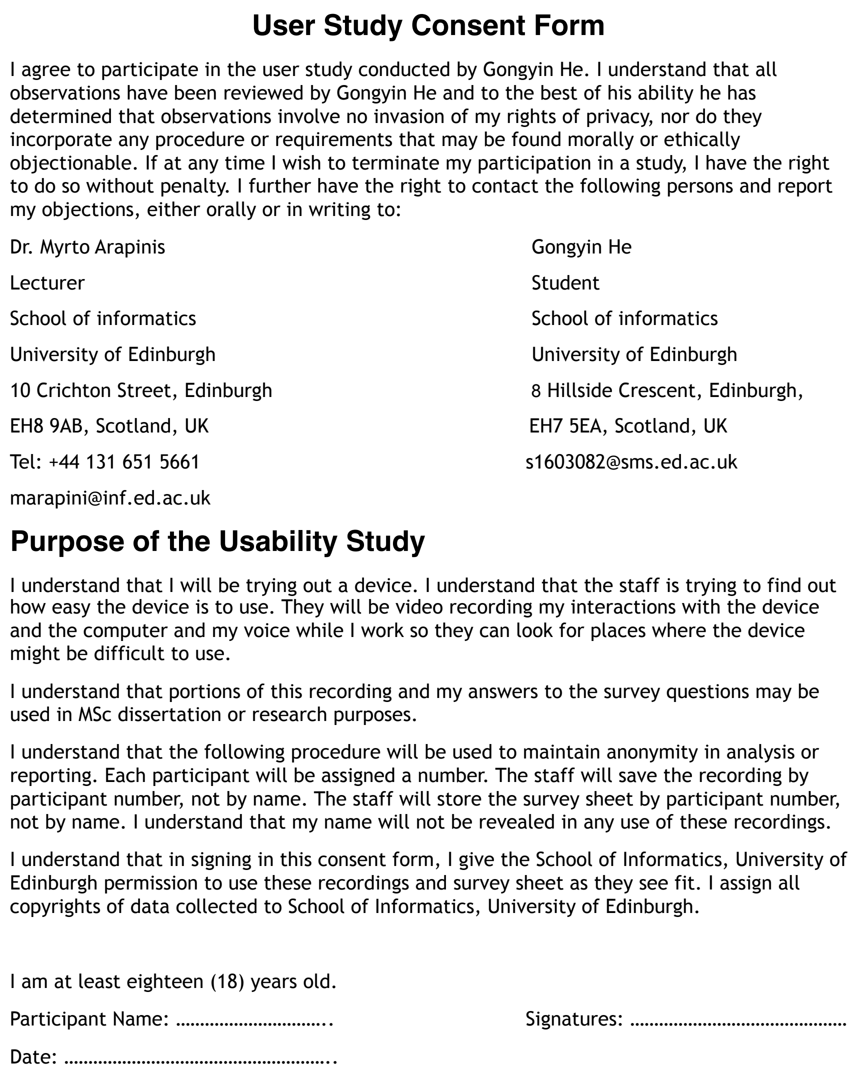
[33] "random — android developers," https://developer.android.com/reference/java/util/Random.html, 2017, [Online; accessed 6-April-2017].

[34] "securerandom — android developers," https://developer.android.com/reference/java/security/SecureRandom.html, 2017, [Online; accessed 6-April-2017].

[35] H. Krawczyk, “How to predict congruential generators,” *Journal of Algorithms*, vol. 13, no. 4, pp. 527–545, 1992.

Appendix A

Consent Form



User Study Consent Form

I agree to participate in the user study conducted by Gongyin He. I understand that all observations have been reviewed by Gongyin He and to the best of his ability he has determined that observations involve no invasion of my rights of privacy, nor do they incorporate any procedure or requirements that may be found morally or ethically objectionable. If at any time I wish to terminate my participation in a study, I have the right to do so without penalty. I further have the right to contact the following persons and report my objections, either orally or in writing to:

Dr. Myrto Arapinis	Gongyin He
Lecturer	Student
School of informatics	School of informatics
University of Edinburgh	University of Edinburgh
10 Crichton Street, Edinburgh	8 Hillside Crescent, Edinburgh,
EH8 9AB, Scotland, UK	EH7 5EA, Scotland, UK
Tel: +44 131 651 5661	s1603082@sms.ed.ac.uk
marapini@inf.ed.ac.uk	

Purpose of the Usability Study

I understand that I will be trying out a device. I understand that the staff is trying to find out how easy the device is to use. They will be video recording my interactions with the device and the computer and my voice while I work so they can look for places where the device might be difficult to use.

I understand that portions of this recording and my answers to the survey questions may be used in MSc dissertation or research purposes.

I understand that the following procedure will be used to maintain anonymity in analysis or reporting. Each participant will be assigned a number. The staff will save the recording by participant number, not by name. The staff will store the survey sheet by participant number, not by name. I understand that my name will not be revealed in any use of these recordings.

I understand that in signing in this consent form, I give the School of Informatics, University of Edinburgh permission to use these recordings and survey sheet as they see fit. I assign all copyrights of data collected to School of Informatics, University of Edinburgh.

I am at least eighteen (18) years old.

Participant Name: ………………………….. Signatures: ………………………………………

Date: ………………………………………………..

Appendix B

Think Aloud Script

B.1 Beginning

Hello, my name is Gongyin He. Today we will be using PassSafe on an Android phone to do some simple tasks to test its usability. PassSafe is a Password Manager Application which can help users manage their login information with just a master password. It can also generate secure passwords for them, and automatically fill password on other third-party apps on the device. Meanwhile, we will have a mock-up app which only has a sign-in and register interface to coordinate with the test. You will not be entering any personal confidential information and your participation today is purely voluntary, you may stop at any time. The purpose of this exercise is to identify issues with the Password Manager app. Please remember we are testing the app, we are not testing you. If you agree to participate, (give them the consent form) you may sign this consent form, which the University requires me to use. If you sign it, it means you are giving me the permission to use task results for analysis, and it tells you whom to contact if you want to report any objections. I'll give you two copies - one is for you to keep, and the other is for you to sign and return to me.

B.2 Think aloud training

During this process, you need to talk aloud when you are dealing with the tasks. It means that I would like to know what you are thinking in details as you conduct these tasks. I hope that you can speak out and keep talking aloud during the whole process. If you forget to talk, I will remind you. Meanwhile, please just pretend to be alone and do not plan to a demonstration or give any explanation about your narration. Additionally, I would be grateful if you can ask questions and I will answer the questions at the end of this process. By far, do you have any questions?

Now I have some tasks printed out for you. I am going to go over them with you and see if you have any questions before we start. [Hand them the task.] Here is the task you will be working on. Why dont you read it aloud just so you can get comfortable with speaking your thoughts?

Do you have any questions about the task?

You may begin.

B.3 Tasks

B.3.1 Task 1

Find and open the PassSafe, and follow the instructions to set a master password. You can use **Qwer123-** or anything you like. When the app has been initialised, close it and open it again. Try to login in it.

B.3.2 Task 2

Add a new password note with the given account information below, and use the password generator to generate a random password for this note.

Mock-up App Account

Username: participant

Password: [generated password]

B.3.3 Task 3

Open and view the note you just created. Change the password to **Password**. Finally, delete this note.

B.3.4 Task 4

Enable the autofill functionality, and use the autofill to sign in the Mock-up App.

B.3.5 Task 5

Register a new account in the Mock-up App with the given information below and use the Quick Save function to store the registration information into PassSafe.

Mock-up App Account

Username: newuser

Password: [generated password]