Graz University of Technology
IAIK
Institute for
Applied Information Processing and Communications
Inffeldgasse 16a
A-8010 Graz

Bachelor’s Thesis

Android Bluetooth Credential Store

Camilla Reis
(01430500)
c.reis@student.tugraz.at

Supervisor
Dipl.-Ing. Dominik Ziegler

Graz, 27th August 2018

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The document uploaded to TUGRAZonline is identical to the present thesis.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Dokument ist mit der vorliegenden Arbeit identisch.

Date/Datum	Signature/Unterschrift

Abstract

In this paper, we explore a more secure way to share credentials over multiple devices without relying on cloud services. Sensitive data is stored on a mobile phone while accessing them over a secure Bluetooth connection.
It features an Android application for mobile phones and a Chrome web extension which uses the Web Bluetooth API for establishing a Bluetooth Low Energy (LE) connection between device and website. The application acts as a peripheral and advertises the data. The web extension serves as the central and reads the data after a successfully established connection. The extension then fills the read data into the login fields of the webpage.
To securely store sensitive information on the mobile phone it is encrypted using the symmetric-key algorithm Advanced Encryption Standard (AES) with a 128bit-generated key. The encrypted credentials are then stored using the greenDAO ORM which manages all operations of the database. The generated key lies in the Trusted Execution Environment (TEE) which is handled by the Android KeyStore and represents the most secure option for this project.
Assurance of authenticity of the user is reached by making use of biometric identification respectively the user's fingerprint to authenticate the sharing of the credentials.

Keywords— *android, password manager, mobile application, aes symmetric encryption, bluetooth low energy, web extension, web bluetooth api*

Contents

1 Motivation **6**

2 Related Work **6**

3 Architecture of Project **6**
3.1 Activity *Accounts* 8
3.2 Activity *Connection* 9
3.3 Storage of Credentials using the greenDao ORM 11
3.4 Encryption of Data 12
3.5 Hardware-backed Keystore 13
3.6 Authentication through Biometrics 13
3.7 Peripheral and Central roles 15
3.8 Using the Web Bluetooth API with Chrome extension 15

4 Security Evaluation with Common Criteria **15**

5 Bluetooth and it’s Security **15**

6 Limitations of Application **15**
6.1 Bluetooth LE instead of Bluetooth Classic 15
6.2 Security decreases with root privilege 16

7 Future Work **16**
7.1 Bluetooth Classic instead of Bluetooth Low Energy 16
7.2 Encryption of BLE connection 16
7.3 Encryption of database tables 17
7.4 Implementation of password generator 17
7.5 Creating accounts from the broswer 17

8 Conclusions **17**

List of Figures

1 MainActivity 7
2 Activity *Accounts* 9
3 Activity *Connection* 10
4 Activity for Authentication 14

Listings

1 CredentialApplication.java; Creation of database 11
2 DBHelper.java; Insert entry into database 12

Introduction

Nowadays password managers are essential to our everyday lives. Studies show that the average person has up to 20 online accounts [paper]. As shown in [ref:typicalpassword], the average user also does not create particularly safe or hard to remember passwords. Often the same password or some variation of it is reused over multiple different accounts. These passwords are not secure and very vulnerable to attacks. [ref]

However, to maintain all account logins, many users rely on password managers to handle it for them. This makes it easier to create and remember multiple passwords, which are different from each other but at the same time equally safe. There are different kinds of password managers. Some store the data locally, for example on a computer or mobile device. This is a safe option to manage credentials but making them accessible throughout different devices can be challenging. In order to make them available on multiple devices, some of those password managers offer the possibility of cloud synchronization. This may be very convenient when working on different devices but not very reliable when it comes to the security of data. The risks are data being corrupted, stolen or wiped. [https://www.bbc.com/news/business-36151754] [paper attack on cloud services]. Then there are password managers that are entirely cloud-based meaning that no data is saved locally on any device and credentials are accessed over a website. Also, some password managers support browser integration which makes passwords accessible through the browser in use. By uploading sensitive data in any form onto the internet, the risk of an attack may increase. [ref] The goal is to store credentials as safely as possible and to make them available on different devices while at the same time without relying on any cloud services.

In this project, an Android credential manager is designed that does not store data onto the internet but rather shares them via a secured Bluetooth Low Energy (LE) connection established with a Chrome extension using the Web Bluetooth API [ref web Bluetooth API google link]. The data is stored on the device and only on demand one selected credential pair consisting username and password is shared with the extension at a time. Accessing and sending credentials requires authentication from the user in the form of biometric information or a master password. It is guaranteed that only the owner of the mobile phone can access and share his credentials. The Web Bluetooth API [chome API link] on the other hand receives the data that is sent from the Android application and fills them in into the text fields on the website. Neither is it necessary to type in the credentials manually nor even make them visible.

This paper consists of 8 chapters that are structured as follows: Section 1 shows the motivation behind this project, while section 2 explains related work that has already been done on this subject. Section 3 handles the architecture of the project and presents the topics in detail. Section 4 illustrates a brief security evaluation with common criteria, and section 5 discusses Bluetooth, and it's underlying security. Furthermore, section 6 describes the limitations that apply to this project and future work is explained in section 7. Finally, the conclusions are drawn in section 8.

1 Motivation

The motivation of this project is to provide a safe possibility for mobile phones to store credentials that are available to the user on multiple devices without cloud storage.

There exist many applications that offer the service to store user credentials safely. To synchronize data between multiple devices they rely on cloud services where data confidentiality cannot be guaranteed. Mobile phones offer a secure way to store data, for example through hardware-backed key storage. By relying on this technology, we can provide secure data storage for user credentials.

2 Related Work

papers on similar topics

PWM local w/ possible cloud sync PWM cloud based both types support browser integration

Or just browser itself offers to store credentials for the next time loging into the website.

password managers that rely on cloud services

password managers that are integrated into web service, google pw managaer, firefox, ...

security issues e.g. last pass had leaks in their release.

MAYBE put to related work which decisions we made, which cipher, which encryption method, ...??

3 Architecture of Project

As mentioned above, we decided to implement an Android application for mobile phones to store sensitive data on it. Since the Chrome browser offers an interface for using Bluetooth LE on the web, we decided to develop a Chrome extension to utilize their Web Bluetooth API. [link]

The API initiates the connection upon a button click from the user and searches for BLE devices that are near and offer a specific service that is defined through a UUID, a universally unique identifier. Once paired with the mobile app, the secure Bluetooth connection has been established, and the transmission of data can take place. Upon successful pairing, the extension is allowed to read characteristics that are advertised on the app. The characteristics contain the values of username and password, which the extension inserts directly into the login fields of the website.

explain more about web bluetooth api, technical details, and code needed to work witht the BT api,
First, the requirements of the application as well of the extension were defined:

Application requirements:

- Safe storage of credentials (account name, username, and password)
- Adding, changing and deleting credentials

- En- and decryption of data
- Authentication through biometric identification
- Establishing a Bluetooth connection with the web extension

Extension's requirements:

- Injection of a button onto the website to establish a Bluetooth connection
- Reading the advertised characteristics
- Filling credentials into the proper login fields

For convenience purposes, we implemented the application for API level 23 minimum and up using Android Studio as the IDE. To go more into detail about the app, it consists of a MainActivity.java [ref figure 1], which contains four different menu items: *Accounts*, which provides the user with an interface to manage their login accounts; *Connection*, where users can establish the Bluetooth connection and send their login data to the web browser; *Settings* and *About*. We will go more into detail about the functionality of the activities *Accounts* and *Connection*. Subsequently, we will examine how the application handles data storage, encryption, and authentication in greater detail.

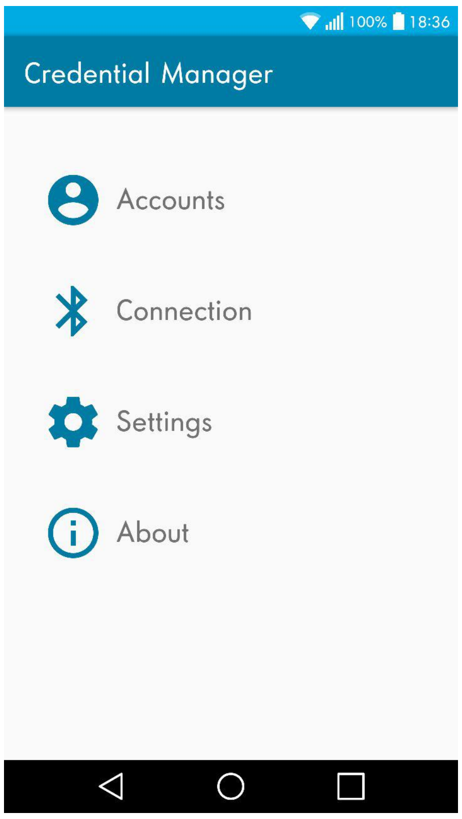
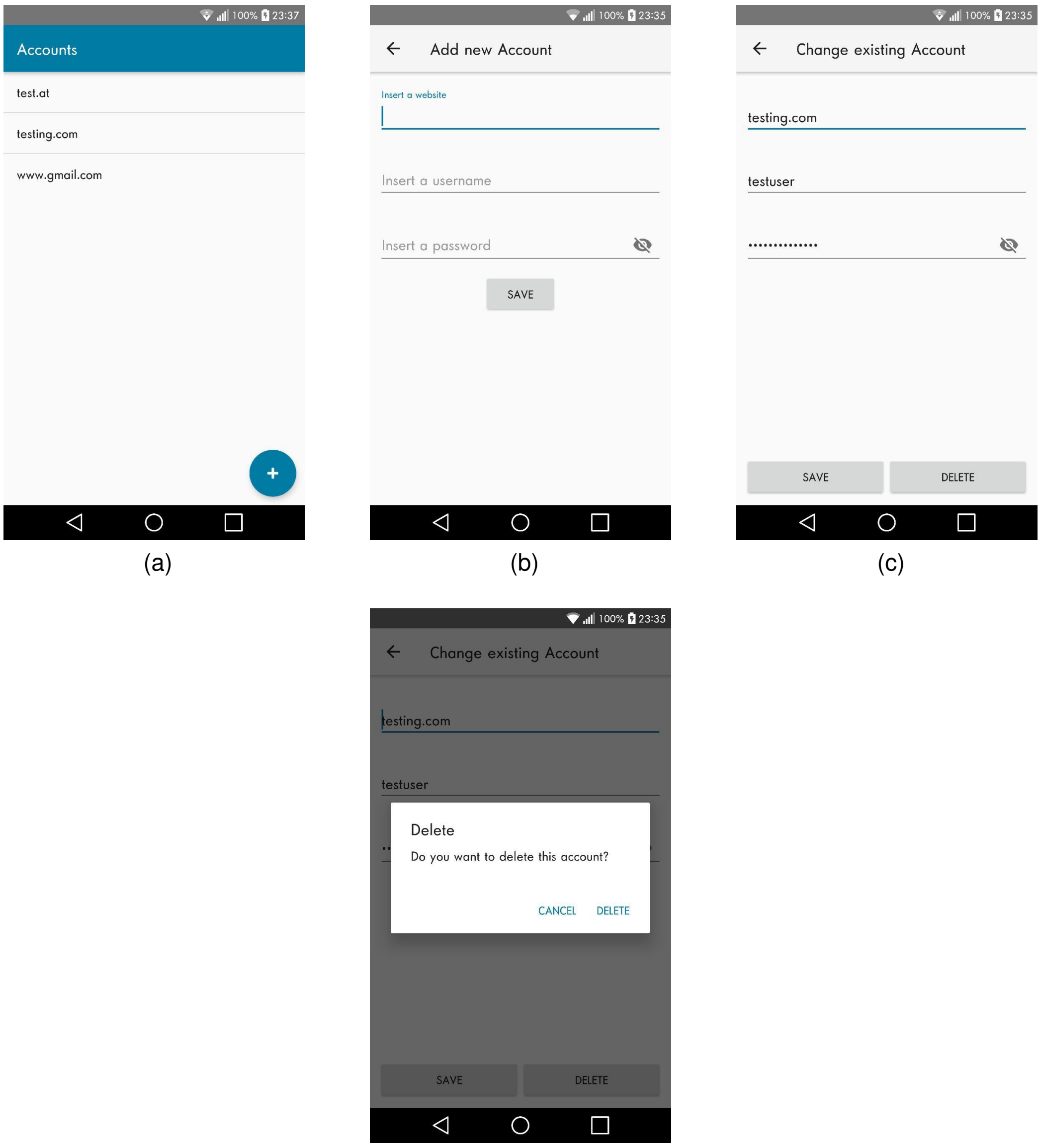


Figure 1: MainActivity, which is shown at start up of the app

3.1 Activity *Accounts*

By clicking on the menu item *Accounts*, the user is asked to authenticate themselves using biometric information in the form of fingerprint or a master password as seen in figure 4(a). Upon successful verification, the interface is shown where the user has an overview of his saved credentials as seen in Figure 2(a). By clicking the floating action button on the bottom right of this interface, the activity is switched to the *AddAccountsActivity*, where the user can add a new login account as seen in Figure 2(b). Here the user types in the website where credentials belong to as well as the username and password itself. Clicking the *SAVE*-button encrypts and adds the credential to the database.

When clicked on one of the existing credentials in the list, it takes the user to the interface shown in Figure 2(c). Here the values can be changed and saved again, or the login account can be deleted from the database. When deleting a credential, the user will be asked via a prompt as seen in Figure 2(d).



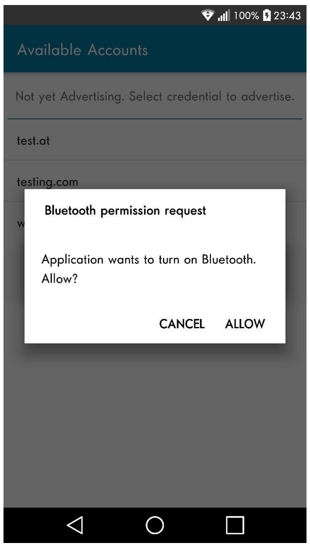
Do you want to delete this account?

Figure 2: Activity *Accounts*: In (a) it shows the listed accounts after success user authentication. (b) is the interface where the user can add accounts and in (c) the user can change a selected account. (d) asks the user before deleting credentials out of the database.

3.2 Activity *Connection*

After selecting the menu item *Connection*, the app checks immediately if Bluetooth is enabled and if not it prompts the user to allow turning on Bluetooth on the mobile phone as shown in Figure 3(a). If the activation is denied, then the application will return to the main activity with an error message saying "*Bluetooth is Not Enabled*" which is shown in Figure 3(b).

Once Bluetooth is enabled, the accounts, which are available to share over the BLE connection to the web browser, are listed. At first, the application does not advertise. The user is asked to select an account that he wants to send to the Web Bluetooth API for login as seen in Figure 3(c). Before the credentials are shared, the user is asked to authenticate himself again using his fingerprint or his master password. Figure 3(d) shows that only after successful authentication the advertising starts and the Web Bluetooth API searches for BLE devices that are near.

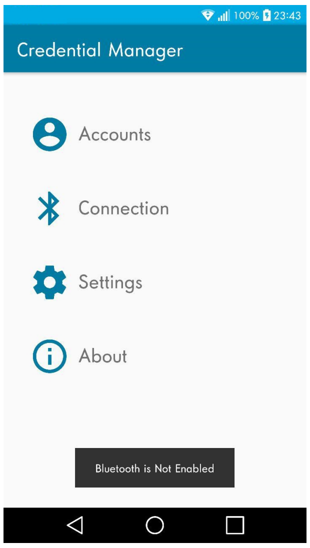


Not yet Advertising. Select credential to advertise.

Bluetooth permission request

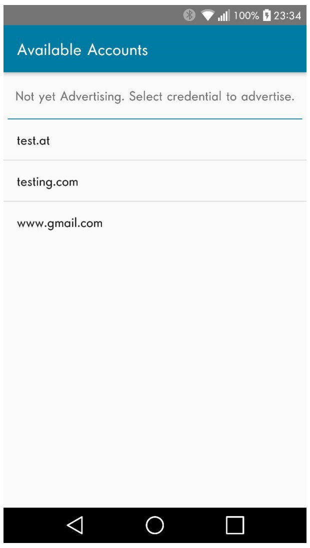
Application wants to turn on Bluetooth. Allow?

(a)



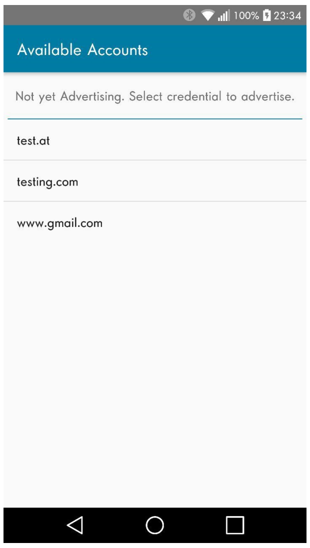
Bluetooth is Not Enabled

(b)



Not yet Advertising. Select credential to advertise.

(c)



Not yet Advertising. Select credential to advertise.

(d)

Figure 3: Activity *Connection*: In (a) a prompt is shown to the user to enable Bluetooth. When the application is not advertising, (c) is shown and (d) is shown after user selected a credential to send.

3.3 Storage of Credentials using the greenDao ORM

Since the goal is to store user credentials on the phone, we need a suitable database structure. One option was using a common SQLite database, which is widely used for Android applications. SQLite databases use, as conventional SQL databases, SQL queries to store, retrieve and delete data. SQL queries are vulnerable to security attacks, for example, second-order SQL injections. As stated in [Detection of SQL Injection Attacks] a SQL injection is defined as a method where code is injected. This exploits a security vulnerability in the database layer of an application. It is done by modifying input data which then alters the SQL query. This increases the risk of storing data safely and may lead to a vulnerable application. In [Detection of SQL Injection Attacks] they state that the risk of a SQL injection attack is higher due to multiple reasons, for example, lack of development time and training, or experience and knowledge of potential security issues.

For this project, we decided to use an object/relational mapping which handles SQL statements in the SQLite database. To manage the tasks of storing, deleting, updating and querying a so-called *object/relational mapping* is used to save time and resources. We decided on the greenDAO ORM [ref http://greenrobot.org/greendao/] as it has an overall good performance and saves time because it does not require as much programming effort compared to other frameworks as discussed in [ref greenDAO paper].

Before we started with setting up the database we needed to included the greenDAO plugin into the project. We added a plugin and a dependency to the app module gradle file, and dependency to the top-level build file. [greendao link]

Next step was the set up the database. This was done by initializing a DaoSession which is given by a DaoMaster. The DaoMaster was provided from the helper class `DevOpenHelper`, which was instantiated in a singleton class of the application as shown in Listing 1.

Listing 1: CredentialApplication.java; Creation of database

```
DaoMaster.DevOpenHelper helper =
        new DaoMaster.DevOpenHelper(this,"credentials-db");
Database db = helper.getWritableDb();
daoSession = new DaoMaster(db).newSession();
```

The DaoMaster handles the database set up and creates DaoSessions. A DaoSession, on the other hand, manages the DAO object, which refers to the entity that we store into the database. DAO objects are created from a Java class with annotations. We used the class `Account.java`, which holds four members: a unique id, the website of the credentials, as well as a username and password. To define this class as a database-backed entity, we added the annotation `@Entitiy` above the class-header. Furthermore, above each class member, we placed annotations as well to define the column names. Afterward, the greenDAO generator automatically generates the constructors, and instances of the class can now be stored into the greenDAO database. [link greendao].

Load, insert and delete a database entry

As mentioned before, the DaoSession manages DAO objects and they, in turn, handle all operations of the database. With the start of the application, the database is opened, and a new DaoSession is instantiated. Additionally, we defined a database helper class `DBHelper.java` where the interactions of DAO objects and the database take place, such as, retrieving all entries or updating a specific one.
Listing 2 shows an example of how one credential set is inserted into the database. The DaoSession calls the DAO object, and the object handles the insert method to store data into the database. The load and delete methods use the DaoSession as well and are similar to the insert method.

Listing 2: DBHelper.java; Insert entry into database

```java
public void insertNewAccount(Account account) {
    AccountDao accountDao = this.daoSession.getAccountDao();
    accountDao.insert(account);
}
```

The database created by the application lies on presistent memory of the mobile device and can only be accessed from our app. Even if the database content is dumped in case of an attack, the attacker only receives encrypted data. In section 3.4 we discuss how data is encrypted before inserting into the database and decrypted again when fetching from it.

3.4 Encryption of Data

For en- and decryption of sensitive data there are some different possibilities. One option of storing passwords is hashing the password and storing that hash instead of the password into the database. SHA256 or SHA512 are some hash algorithms as mentioned in [Implementation and Performance Analysis of PBKDF2]. By hashing data for encryption, data can be recovered by multiple different attacks such as Brute Force or Dictionary attack. Another way to store sensitive data is to hash it with a random string of bits, called a *salt*, which is then saved with the hash of the plaintext. This can also become vulnerable to the Dictionary Attack as stated in [https 3 wrong ways to store a password]. Nowadays it is common to use a key derivation function to safely store passwords such as the *password-based key derivation function 2 (PBKDF2)*, since it provides strong security. Also, *Bcrypt* and *Scrypt* are strong key derivation functions as well. [ref pbkdf2 secure for passwords] PBKDF2 uses the user's input to generate the key for encryption. However, the problem arises that the strength of the encryption will also depend on the password chosen by the user. For various reasons a good master password needs to be chosen but as mentioned above more than 90% of users chose a password that is between 6 and 9 characters long, and most of them do not include non-alphabetical characters, meaning it would take less time to retrieve the password through some attack. [ref 3 wrong ways]. Therefore, we need a way to guarantee safe en- and decryption without depending on the user's ability to create a secure password.

So for this project we decided to encrypt the data using the AES algorithm with a secret key of 128bit length.

The algorithm used to generate the secret key is the AES algorithm. The key is used for encryption as well as decryption. The AES algorithm is a symmetric key algorithm which excrypts data in 128-bit blocks with the generated 128-bit key. [128vs256] We decided to use a symmetric algorithm for various reasons: the algorithm is much faster than an asymmetric algorithm which makes it less expensive but more importantly because there is no need for distribution of a public encryption component. All encryption happens in the background on the phone. So using a asymmetric algorithm such as RSA would be unnecessaryly expensive to compute. As for the mode of the cipher we used a GCM block cipher with no padding.

explain block cipher, explain gcm: deterministic algorithm, operation on fixed length groups of bits called blocks. used aes/gcm/no padding, supported from 23 and above, and supports only 12 byte long IVs change IV so a random string which is created with an own securerandom, explain secure random, link from android website set that keyspecbuilder takes my own created IV with setting (look up in code) this line to "false". this means that android allowed the usage of an own IV instead of relying on their own, since they think the developer is not secure enough (search link that IV cipher get IV is not secure enough –> book). explicitely set the key to use own secure random as the IV.

after the key is built, can be stored away and is retrieved using an alias. an alias is an identifier that refers to the key that is stored in the TEE.

[2] [3]

3.5 Hardware-backed Keystore

As for this project we chose to rely on the hardware-backed Android KeyStore to generate a 256bit key. To create a key that is then used for en- and decrypting data, we implemented an instance of the Android Keystore. In [1] it is stated that the Android Keystore provider lets the individual app store and manage its own credentials while making sure no other application can access them.

the key never leaves the trusted environment and all encryption operations are done in the background.

as mentioned, the key is stored in the Trusted Execution Environment (TEE). The TEE is... paper The TEE is supported by the device manufacturer and represents the most secure option to store the encryption key. the tee is supported by the device. as for all android device that have an qualcomm CPU the tee is supported. paper

explain hardware-backed keystore, and android keystore, that android with qualcomm cpu implements the tee. securest way to store keys, never leaves the environment (paper) what does hardware backed mean. and tee how does the android keystore play its role, only can use android key store if device is secured. meaning if device is looked using a pattern, lock, etc or fingerprint. if device is not locked then it is easy to retreive the .... what? lock screen must be set in order to use key. this is done with the class KeyguardManager. The KeyguardManager ...dev site android cite

3.6 Authentication through Biometrics

To securely protect the access and sending of user credential the app uses biometric authentication in the form of fingerprint scanning. The application does not store any biometric user information into the database. Rather, it retrieves the already saved fingerprints on the

Android mobile phone. This way it ensures that only the owner of the mobile phone has access to the credentials and the app does not have to handle the storage of any fingerprints. In addition the application also checks if a fingerprint sensor is available on the device and if the permission to use fingerprint scanning is granted from the Android permission system. (maybe ref to chap.6.2 more details about permission system).
Before starting authetentication via fingerprint the application makes sure that the lock screen is secured by a PIN, password, pattern or if the SIM card is locked. This is done by the `KeyguardManager` because if provides access to the lock screen. Only in case of a secured mobile device the authentication process will proceed.

The application checks if the saved fingerprints contain the one that the user scanned and returns a successful authentication. Otherwise, it throws an error saying that the authentication failed. If the phone does support fingerprint scanning, but there are no fingerprints stored on the device, the application returns an error message saying that the user must save at least one fingerprint onto the device. If the phone does not have a fingerprint sensor to support scanning mechanism the user has the option of entering a master password, which is set upon the first usage of the application.

As mentioned above, upon every request to send an account login the user has to reauthenticate using his fingerprint or a master password before sharing credentials with the Web Bluetooth API. This security mechanism would prohibit unwanted distribution of sensitive data if the user left their phone unattended with the phone unlocked and the application still running.

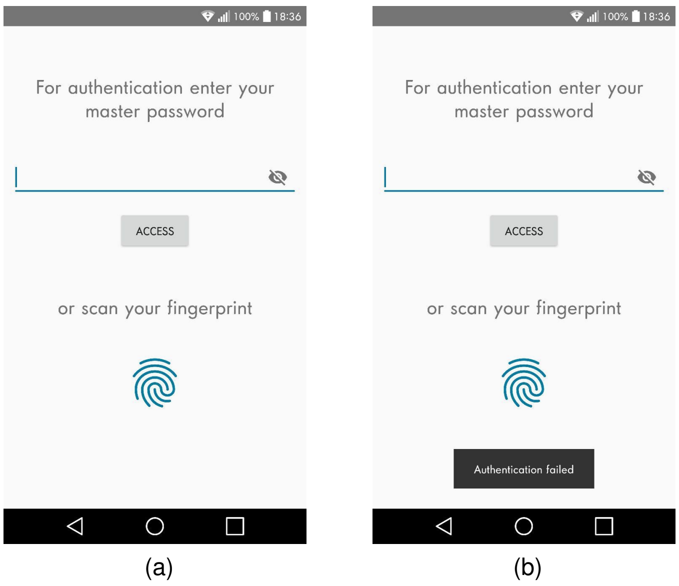


Figure 4: Activity for Authentication: In (a) the user is asked to auhenticate themselves using fingerprint or a master password. In (b) authentication failes if incorrect fingerprint is scanned.

3.7 Peripheral and Central roles

After discussing in depth the credential storage, encryption and the authentication of the user, we will briefly describe the roles of the extension and the Android application.

In common use cases, the Android application has the central role in a Bluetooth LE connection, for example when communicating with Bluetooth LE devices, such as a heart rate monitor or beacon. Therefore, the app serves as the client while the Bluetooth LE device acts as the server, sending data that the application wants to receive.

In this scenario, however, the roles are reversed. The Android app serves as the server, where it advertises the data that the Bluetooth LE extension wants to read. Consequently, the application opens a GATT server to advertise the characteristics and the extension, on the other hand, acts as the client and retrieves those advertised characteristics from the app. The extension implements a GATT service and reads the values from the characteristics, which are the login credentials.

3.8 Using the Web Bluetooth API with Chrome extension

4 Security Evaluation with Common Criteria

see how to conduct a security evalutation of an android app, list all common criteria, maybe make a table??

5 Bluetooth and it's Security

intro with bluetooth classic and for what it is normally used. large data, consistent connection communication channel. but secure.; paper security about bluetooth classes. ref section limitations that we cannot use BT classic because of API. we chose ble because of the chrome web bluetooth api.

so we use BLE: what is BLE, how is it differenct from bt classic and for what it is used, list all advantages and how it conributes to this project. general structure of bluetooth low energy, common usage and applications nowadays. doesnot need much energy, no consistant communication channel less battery. we only advertised upon selection of credentials and once they are sent, the app does not keeps advertising and the user has to select an other credentials and authenticated themselves. for our purposes ideal but, explain about the protcol stack is built differently compared to bt classes. different protocols makes it more insecure [paper with low energy, come low security]

6 Limitations of Application

In this section, some encountered limitations are discussed since they have not been established yet due to necessary technological progress of components or infeasibility.

6.1 Bluetooth LE instead of Bluetooth Classic

For the connection, we chose Bluetooth LE since the Chrome Web Bluetooth API can only support Bluetooth LE at the moment. Although as mentioned earlier, Bluetooth LE can limit the security, it can be a task for future work to encrypt the communication channel further by using asymmetric encryption. ???

more about: bluetooth classic on the other is hand is not supported by the Chrome Web Bluetooth API. therefore it was not possible to implement bluetooth classic. ble classic has a more secure protocol stack [paper] but uses a lot of resources. bluetooth classic mainly used for consistant transfer of data rather than a single transfer of small pakets. BT classis is normally used for audio streaming of sending videos, pictures and larger data which is devided into blocks. is it requried to have a persistant communication channel, whereas with our application it is not neccessary to keep the connection for longer time than needed. This make the app more vulnerable to attacks.

6.2 Security decreases with root privilege

When the device is rooted, the security of the user credentials cannot be guaranteed anymore. As stated in [oncerootalwaysthreat], the Android permission system is one of the most important security mechanism for protecting critical system resources on Android phones.
Before an application can access resources, it needs certain permissions which are granted by the Android permission system. The application asks for permission with check functions and in case of approval the API returns `PERMISSION_GRANTED`, or in case of denial, it returns `PERMISSION_DENIAL`.
Root privilege on an android phone is the highest privilege in user mode and does not need any further permissions from the system to gain access to sensitive data or hardware interfaces, which leads to high risk in security when storing private data. Hence, Android's Shared Preferences and SQLite databases become accessible as mentioned in [Vulnerability Detection in Recent Android Apps]. Therefore, it is not possible for rooted devices to provide a safe possibility to store user credentials.

7 Future Work

This section presents ideas that could be actualized in the near future once the necessary technology has progressed. These features are not part of the scope of this project and therefore have not been implemented further since they may be quite time-consuming to realize or not possible with start-of-the-art technology.

7.1 Bluetooth Classic instead of Bluetooth Low Energy

If in the future the API also supports the implementation of Bluetooth classic, maybe switch to BT classic?
maybe more secure, no advertising. maybe better for sending back information? –> research

7.2 Encryption of BLE connection

Encrypt Bluetooth LE connection of phone and web API with own encryption keys for a safer communication channel. BLE unsafer [ref paper with low energy comes low security], paket sniffer can read out all data in plain text that is sent over a ble connection.

7.3 Encryption of database tables

Encrypt the tables with bcrypt to provide higher security. although key cannot be retrieved from the TEE to decrypt the data in the database, but to create a safer environment it is possible to decrypt the entire table with scrypt/bcrypt.

7.4 Implementation of password generator

Implementation of a password generator in the application to increase the convenience for users to create safer passwords. generate passwords that use special characters, letters and numbers and that are more than 20 characters long.

7.5 Creating accounts from the broswer

Create new login credentials and send them back to the application for storage. not only sending password to the website but also while creating a new account for a website user can select an option to send those newly created credentials back to the application to securely save them on the device.

8 Conclusions

Short summery of project, 300-400 words

References

[1] “Android keystore system.” `https://developer.android.com/training/articles/keystore`, 2018. [Online; accessed 06-08-2018].

[2] Z. Ahmad, L. Francis, T. Ahmed, C. Lobodzinski, D. Audsin, and P. Jiang, “Enhancing the security of mobile applications by using tee and (u) sim,” in *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pp. 575–582, IEEE, 2013.

[3] T. Cooijmans, J. de Ruiter, and E. Poll, “Analysis of secure key storage solutions on android,” in *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pp. 11–20, ACM, 2014.