Password Storage

Eamon O'Brien

March 7, 2020
Version 0.2

1 Summary

Many end-users for mobile surfaces appreciate simplicity - yet in contrast the requirement for password protection is the opposite. Passwords require more and more complexity as time goes on, and with computational power continually advancing these passwords get even more difficult to remember, especially with the common requirements for special characters, upper- and lower-case letters, and numbers. This program aims to encourage the use of longer, more complex passwords that are difficult for both humans and computers to decode by removing the need for a user to remember all except one password, which could be replaced with alternative means such as thumbprint recognition.

1.1 Confidentiality

Since storing passwords is a risky move for many reasons, this project aims to put minds at ease for issues such as third party penetration and personal privacy by encoding all data stored coupled with storing all data on the local device. No data pertaining to the identity of the user, nor the details they have entered to this application will be stored by ¡Corporation¿ unless explicity authorised. ¡Corporation¿ takes confidentiality seriously, and believes that the internet must be a place that no country or corporation may own.

1.2 Tim's Mobile Games

Idea: a knock-off of Jim's Mowing and the like, could be a similar guy using a keyboard with glasses on or something lmao.

2 Part 1: Command Line Interface

2.1 User

A user is capable of creating an account, or logging into an existing one, with a "master password". This password is used to decrypt all of their data, while a public key saved with their account is used to encrypt it. All data is stored by a third party SQL based application, but in an encrypted form to ensure no risk of third party requests accessing data. When a password is requested, it should be copied to the clipboard.

2.2 Database

The database storing passwords needs to be accessed and managed by this application, with tables being successfully created and managed by the application. Test cases should create similar tables to mirror the database as lists for the majority of cases, while hard tests of the database will create testing tables to be created and then dropped in the same instance to minimise data. All data entries should be sent over network sockets in an encrypted form, and decrypted by the main application.

TODO: can I store the data decrypted within the program?

2.3 Password Generation

While a lot of people already have passwords, they often tend to be mirrors (or exact duplicates) of each other, thus generating passwords and then storing them in the application would be of immense use to a lot of people. These passwords should be sufficiently complex, but also memorable.

2.4 Security

The application should be sufficiently secure, consult outside sources for likely problems that have not already been addressed.

2.4.1 Database Access

As above, all data should be encrypted to prevent third party access

2.4.2 Local Storage

All data stored locally should be in hash-code forms, and any printed data should be wiped.

NOTE: is storing the data and decrypting it very often (requiring password each time) effective for UX?

3 Part 2: Graphical User Interface

Although the command line is popular for Linux users, the more common front is for mobile users, which in itself requires a more complex environment that text. i.e. most people hate text, so we need a prettier environment.

3.1 Entrance

The user should be greeted with a nice pretty page about who wrote this program (me), like all apps have.

3.2 Login

Since the command line requires a user to log in, the user should enter a username and password combination. Should it be an existing combination, they may then view what entries they have on the next page.

3.3 Entries

Most likely with a drop-box or similar, the user must be able to view what entries they have available [this is actually where decoding it all to plaintext might be an advantage]. They should then be able to view the details they've entered for an account, and add, modify, or delete the entry. If it's the entry they're looking for, they should also be able to view or copy (we'll see what we/they prefer) the associated password for use.

3.4 New/Edit Entries

Should the user wish to update an entry or create a new one, a heap of fields should show up for them to enter data into. It should then be encrypted and the database updated to ensure their next request is good.

4 Style

The primary style here will be based on two components: interaction with internal components, and interaction with external components. This follows with many sections of the program, including user input interaction, UX, and database inputs.

4.1 Require versus Parameter

Two independent styles of Java commenting are the Require/Ensure and Parameter/Return pairs.

4.1.1 Require/Ensure

The require and ensure pair will be used internally where it can be fairly assumed that no null or incorrect values will be present, due to no interactions with exterior inputs such as databases or general user input. Since they essentially lower the volume of code required, they are very helpful when writing larger stacks and will be used in a lot of areas of this project (under the assumption that they are handled and called correctly).

4.1.2 Parameter/Returns

The classic parameter and returns (with optional throws) commenting style generally takes more space, both within the commenting and within the code style due to requirements to check for these potential problems and restrictions. Because of this, common sense can often replace these problems inside the main code, but external interactions require filtering. These cases will also include primary bases for classes such as RSA algorithms that require checking: these checks may however be assumed correct at higher levels.