Table 1: Revision History

Date	**Developer(s)**	**Change**
November 10, 2017	Thomas Mullen	Rev.0 of Document
...	...	...

3XA3 Design Document

Group 20 (2020Vision)
Mullen, Thomas - mullentc - 001406837
Pavlich, Phillip - pavlicpm - 001414960
Bauer, Ivan - bauerim - 001418765
L02

Contents

1 Introduction and Overview

1.1 Summary of Project

Password security is a persistent issue that affects all users who need to protect sensitive data and systems with passwords. Users often struggle to remember secure passwords due to their length and end up choosing short passwords that offer reduced security.

A common solution to this problem is to use a "password manager", a piece of software that securely stores passwords. The most common implementation of a password manager is an encrypted database of passwords stored locally on the user's system. These stores are vulnerable to decryption by brute force, or through flaws in the encryption algorithm.

This document describes the design of a program (referred to as PretzelPass) that uses an alternative method. This method uses a hashing algorithm to generate passwords when they are needed, as well as a securely stored salt, to make brute-force attacks computationally infeasible.

1.2 Design Principles

The development team have followed a structural design pattern called the bridge pattern. The bridge design pattern consists of separating an object's interface from its implementation. We did this by splitting the code into three different modules called browseraction, hasher and options. The user interface is in the browser_action directory. The algorithmic operations to obtain and modify the password are separated into a module called hasher. Options holds all of the user's password generation preferences and display preferences. To reduce coupling among the modules, inter-module communication is managed through event subscriptions.

The behavioral design pattern that we have used is called a state design. We have used a series of event listeners to recalculate every time that a state on the front end has been modified. This allows the system to automatically update itself without prompting the user. This increases the speed and efficiency of the program.

1.3 Document Context

This document should be examined in context with the Module Interface Specification (MIS) and the Software Requirements Specification (SRS).

1.4 Document Structure

The structure of this document is detailed in the table of contents.

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

The software may encounter future issues that cannot be dealt with until they occur. The designers acknowledge several potential issues could arise and have modified the design to compensate for these possible changes. Anticipated changes shall only affect a single module in order to minimize future maintenance costs. This will also prevent the integrity of the product from being compromised as a whole. Anticipated changes for PretzelPass are as follows:

- AC1) The hashing algorithm used to secure passwords may need to be updated to a newer, more secure version.
- AC2) Increase in computing power may require multiple rounds of hashing to be used.
- AC3) Code that interacts with the Chrome Extension APIs may need to be updated as these APIs are removed or updated.

2.2 Unlikely Changes

While there are many aspects that are likely be required to be changed in the future, there are also several aspects of PretzelPass that are unlikely to need any changes. Unlikely changes for PretzelPass are as follows:

- UC1) Password-based services are phased out and the software needs to be deprecated.
- UC2) Generating passwords through hashing algorithms is shown to be insecure and the generation method must change.
- UC3) The medium that the extension is provided through (Chrome) becomes deprecated and a new browser must be adopted.

3 Module Hierarchy

The modules will be split into three main modules, each containing their own sub-modules. This design supports information hiding.

- M1) Browseraction Module
  - M1A) UserInterface
  - M1B) Customization
  - M1C) EventListeners
- M2) Hasher Module
  - M2A) index
  - M2B) hash
  - M2C) unicode
  - M2D) constraints
  - M2E) salt
- M3) Options Module
  - M3A) constraints
  - M3B) display

Table 2: Module Hierarchy

Level 1	Level 2
browseraction	UserInterface Customization EventListeners
hasher	index hash unicode constraints salt
options	constraints display

4 Connections Between Requirements and Design

The design of the project implements the requirements detailed in the Software Requirements Specification. The following Traceability Matrix details this relationship.

5 Traceability Matrix

Table 3: Trace Between Requirements and Modules

Requirement	**Module**
R1	M2B
R2	M1C, M2A, M2B,M2C, M2D, M2E
R3	M1A, M1B
R4	M1A, M1B, M1C, M3A, M3B
R5	M1C
R6	M1A, M3A
R7	M2A, M2B,M2C, M2D, M2E, M3A,M3B
R8	M2E
R9	M2A, M2B, M2C, M2D, M2E
R10	M2E
R11	M1A, M1B
R12	M2E
R13	M1A, M1B, M2A, M2B, M2C, M2D, M2E, M3A, M3B
R14	M2A, M2B, M2C, M2D, M2E
R15	M2A, M2B, M2C, M2D, M2E
R16	M2A, M2B, M2D, M2E
R17	M1A, M2A, M2B, M2C, M2D, M2E, M3A, M3B
R18	M1A, M2A, M2B, M2C, M2D, M2E
R19	M2A, M2B, M2D

Table 4: Trace Between Anticipated Changes and Modules

Anticipated Change	**Module**
AC1	M2B
AC2	M2B
AC3	M1A, M1B, M1C, M2A, M2B, M2C, M2D, M2E

The table above shows what modules would be affected by the anticipated changes outlined in Section 2.