Computer Science NEA

Password suite

Contents

[Analysis 3](#_Toc1422580)

[Introduction 3](#_Toc1422581)

[Identifying the End User 3](#_Toc1422582)

[Other Prospective Users 4](#_Toc1422583)

[How Accounts Are Compromised 4](#_Toc1422584)

[The Current System 4](#_Toc1422585)

[Features Required by the Client 5](#_Toc1422586)

[Degree of Encryption 5](#_Toc1422587)

[Proposed Solution 6](#_Toc1422588)

[Objectives 6](#_Toc1422589)

[Password Manager 6](#_Toc1422590)

[Password Generator 6](#_Toc1422591)

[Password Tester 7](#_Toc1422592)

[Data Flow Diagram 7](#_Toc1422593)

[Evidence of research 8](#_Toc1422594)

[Microsoft Research 8](#_Toc1422595)

[Popular Password Managers Pricing 8](#_Toc1422596)

[Interview with Client 9](#_Toc1422597)

[Common Password Requirements 10](#_Toc1422598)

[Google 10](#_Toc1422599)

[Amazon 10](#_Toc1422600)

[Spotify 10](#_Toc1422601)

[Documented design 11](#_Toc1422602)

[Overall Design 11](#_Toc1422603)

[Modular Design 13](#_Toc1422604)

[Data storage 13](#_Toc1422605)

[File structure 13](#_Toc1422606)

[Entity Relationship Diagram 13](#_Toc1422607)

[Entity Descriptions 14](#_Toc1422608)

[SQL 14](#_Toc1422609)

[Create tables 14](#_Toc1422610)

[Login/create new user 14](#_Toc1422611)

[Get passwords from manager associated with userID 14](#_Toc1422612)

[Data volumes 14](#_Toc1422613)

[Algorithms 15](#_Toc1422614)

[Encryption 15](#_Toc1422615)

[Hashing 16](#_Toc1422616)

[Login 16](#_Toc1422617)

[Password generation 16](#_Toc1422618)

[Word based generation 18](#_Toc1422619)

[Password Testing 18](#_Toc1422620)

[Testing 21](#_Toc1422621)

[Test stratergy 21](#_Toc1422622)

[Test data set 21](#_Toc1422623)

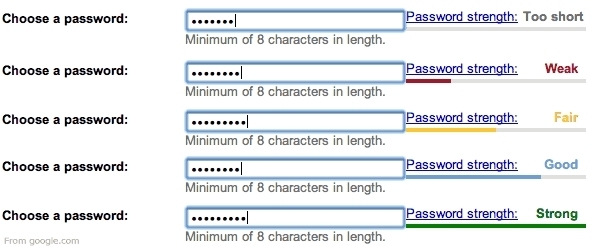
# Analysis

## Introduction

Passwords are often the only means of stopping unauthorised users accessing an account. For this to be an effective scheme, the passwords must be something that only the owner of the account can know. It is advised to use a unique password for every account however, most users find this difficult as remembering many unique passwords, all with different criteria, can be difficult. This leads to people using the same passwords multiple times, which is a security risk since the breaching of one account could lead to all of a user’s accounts being breached.

Moreover, there are many common passwords such as “password” which can be easily guessed by a hacker using a dictionary attack. This dictionary would contain a large selection of passwords, one of which may be the user’s. If the hash of the password was acquired, then a rainbow table would be used to find the password associated with the hash. A rainbow table is a pre-computed list of hashes of common passwords.

Common passwords, and their variations (p@$$wOrD etc.), will all be present in these lists; this makes it trivial to gain access to the account.

Even if people are using unique passwords for each account, that they think are uncommon, there is no guarantee that this password does not appear in tools used by hackers. Often the only means of testing a password’s strength that a user encounters is during the sign-up process, where meeting the sites password requirements will qualify the password as strong even if it is common and can be easily guessed.

## Identifying the End User

The end user for my application is Rylan. He has accounts on many websites with different password policies which he finds difficult to remember. To be able to meet various requirements, he has a small selection of passwords that he uses on multiple accounts. He currently uses the built-in password manager on his mobile devices since it is unlikely that other people will have access to it. However, he regularly leaves his PC unattended therefore, he cannot use the integrated password managers in browsers or keep his accounts signed in as anyone would be able to access his accounts. He does not often use PCs other than his own which is mostly uses for gaming and browsing the internet. Overall, he is confident with technology and thinks that the security of the program is more important than the ease of use.

He has found that many password managers only allow local storage of passwords for a monthly cost, which he felt was unjustified. Also, when asked about the strength of his password he was unsure as he had only ever made passwords to meet the requirements of the website he was signing up to and never tested his passwords in a more realistic environment.

## Other Prospective Users

I believe that my client is typical of the average user, if not slightly more security conscious. Therefore, the application that I build will have usefulness outside of my client. According to “A Large-Scale Study of Web Password Habits” by Microsoft Research, most users tend not to use unique passwords, instead they use 6 or 7 passwords on average for around 4 accounts each. The study found that the main method for gaining access to an account was typing a memorised password, copying a password from a piece of paper or through trial and error of all the passwords that they are likely to use. Finally, it found that 1.5% of Yahoo users forget their password every month.

## How Accounts Are Compromised

For the application to be effective in keeping passwords safe, it must protect against all common ways that passwords are found outside of them being stolen from a company’s database. “Do Strong Web Passwords Accomplish Anything?” was another study conducted by Microsoft Research which investigated how the strength of the password could be undermined by acquiring the raw password. It found that the most common ways that this happened was:

1. Phishing

2. Keylogging

3. A brute-force attack on the user’s account (i.e. an attacker knows the userID and tries to guess the password)

4. A bulk guessing attack on all accounts at the institution

5. Special knowledge or access attacks:

* guessing based on information about the user
* shoulder surfing
* console access to a machine where password auto-fill is enabled

The application must be able to generate passwords that make brute force, guessing based on the user and bulk guessing ineffective. Furthermore, it should be able to protect against keylogging by not requiring a user to ever type a password.

## The Current System

Currently, he memorises most passwords. This limits the complexity of the passwords he can use and forces him to use passwords that may contain common words and phrases. Where the consequences of the account being breached are large enough, he writes passwords on a piece of paper. He hides this piece of paper since it would allow anyone to gain access to these accounts. This is insecure because as soon as someone finds the piece of paper, they have access to all the accounts listed on it. Both methods require the user to type the password which leave room for a transcription error. Multiple errors could lead to the user locked out of his account for some time.

## Features Required by the Client

* Encrypted storage of all account data (username, password, data associated with verifying password) inside a database.
* Database is accessed using a master password.
* Functionality to copy and paste the username and password to avoid transcription errors.
* The ability to generate strong random passwords.
* The ability to generate a string of random words to form a more memorable password.
* Testing of passwords in a realistic environment.

## Degree of Encryption

The degree of encryption required for the application would vary based on the consequences of the user’s account being breached. If there was a high risk of a monetary loss, the encryption would have to be very strong but if the consequences were less severe, the encryption could be less strong. Also, since the application is local, it would have to be stolen for

To understand the degree of encryption required for the client, I asked him to fill out the following questionnaire.

|  |  |  |  |
| --- | --- | --- | --- |
| Name of service that the password manager and generator will be used for | What is the consequence of that account being breached? | | |
| Monetary | Personal | Negligible |
| Google (Gmail, YouTube etc.) |  |  |  |
| Amazon |  |  |  |
| Spotify |  |  |  |
| Reddit |  |  |  |
| Facebook |  |  |  |
| Instagram |  |  |  |

The questionnaire showed that there is only one account not having a considerable impact on the user. The client is only planning on using the application on his home PC, which is unlikely to be stolen. So, the encryption needs to sustain at most three hours of brute force attack since the user would likely notice that the PC has been stolen in around that time and then be able to change the passwords. Therefore, simple hashing and encryption algorithms can be used.

## Proposed Solution

I aim to create a standalone local application that stores, generates and tests passwords. The usernames and passwords entered by the user will be encrypted and then stored in a database using SQL statements. This database will be accessed using a master login, the details of which will be held inside a database. The master login details must be held relatively securely.

The password generator must be random with options to generate passwords meeting various criteria requested by websites. The ability to create passwords that are memorable and secure will be needed to allow the creation of passwords that could be memorised and used without access to the application.

Testing the passwords in a realistic environment could be achieved through searching for the password in a word list.

It will run on only Microsoft Windows but will otherwise not require any additional software. Because of this the program will be completely free to use, unlike other products on the market.

## Objectives

### Password Manager

1. Have a system to login to the database using a username and master password.
2. The user must be able to create a master account.
3. Time out period if the wrong password is entered three times.
4. Master passwords must be stored with an appropriate degree of encryption in a database

* Hashed using a secure algorithm
* Salted using a randomly generated string

1. When logged in, the user is presented with all their stored usernames and encrypted passwords.
2. The plaintext password should not be immediately visible.
3. There should be functionality to copy a password to the clipboard.
4. The user must be able to add, remove and edit stored usernames and passwords.
5. The user’s stored usernames and passwords must also be encrypted using a secure algorithm.

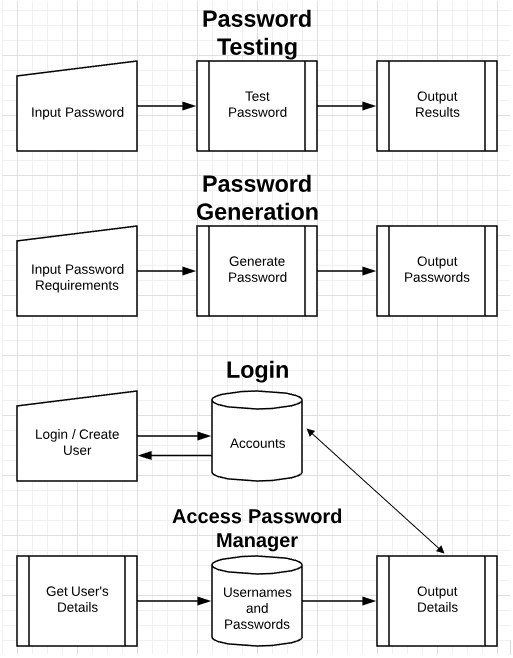
### Password Generator

1. The random number generation must be appropriately secure.
2. Creating random looking passwords including lowercase and uppercase letters, digits and special characters
3. The ability to generate strings of a user chosen number of random words.
4. For the random passwords, the user must be able to input external criteria to match the site they are using the password for such as choosing the length, whether to include uppercase, numbers, special characters etc. using tick boxes.
5. For the string of words, a random number will be generated and the word corresponding to the number will make up the password. This will be repeated for as many words as the user requires.

### Password Tester

1. The user should be able to input a plaintext password, which will then be tested using the following methods:
2. The password will be check against basic rules that websites enforce.
3. The password will be searched for in the provided password dictionary or a user provided password dictionary that can be found online.
4. The dictionary must be searched using a fast algorithm since they often contain hundreds of thousands of passwords.
5. If the password is found to be insecure, the user should be directed to the password generator and given simple tips to improve the password’s strength

## Data Flow Diagram



## Evidence of research

### Microsoft Research

This is the study that I used to get an understanding of how most people use passwords.

<https://www.microsoft.com/en-us/research/wp-content/uploads/2006/11/www2007.pdf>

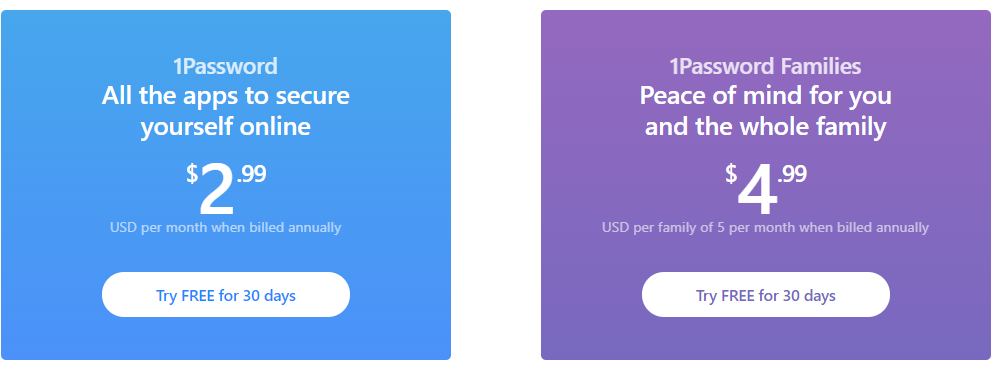
This is the study that outlined the most common ways that passwords are stolen.

https://www.usenix.org/legacy/event/hotsec07/tech/full\_papers/florencio/florencio.pdf

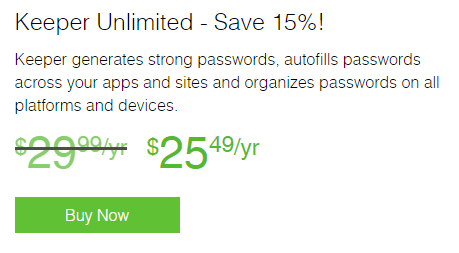
### Popular Password Managers Pricing

Following are the prices of the most popular password managers:

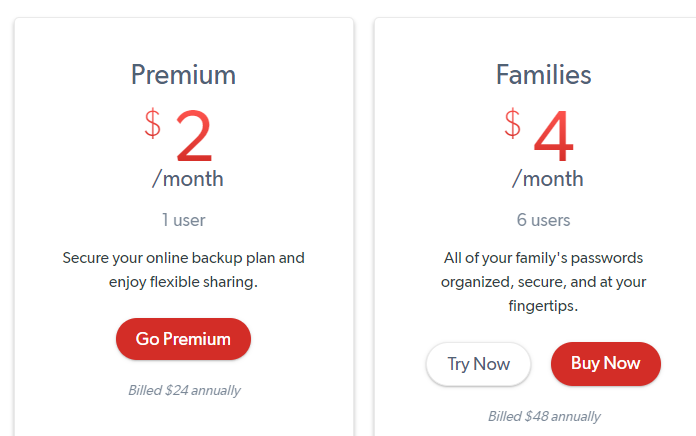
#### 1Password



#### Keeper



#### LastPass



## Interview with Client

To tailor my solution to the client’s specific needs as best as possible, I spoke to him to understand his current system and what issues he would like to be solved.

What is your current password system?

On my PC, I memorise all my passwords and then type them out since I’m not always at it and don’t want anyone being able to access my accounts. But I always stay logged in on my phone.

Do you experience any problems with this?

I often have to type my passwords many times since I make mistakes when typing them. This has led to me being locked out of an account for an hour. Also, I usually have to try a few different passwords until I get the right one for that account.

Do you know the strength of your passwords?

I have only ever seen the strength of my passwords during the signup process when making an account on a new website where my passwords are graded as strong. Other than that, I have never investigated the strength of my passwords, especially not in a realistic environment.

Does your password system lead you to having a password that may be weak?

My passwords tend to be words and phrases that are memorable to me with numbers or characters added depending on the site’s requirements. I would assume that this would be weak since the password is probably quite common.

Have you ever considered using software to manager or generate your passwords?

I have, however, most of the popular applications currently available require you to pay a monthly fee, which I don’t think is worth it and most of the free ones store your passwords on their cloud server which would mean that if they were hacked, all my passwords would be lost.

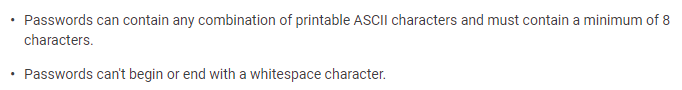
Is ease of use more important than the security of the application?

I am generally quite confident in using software and since I won’t need to access the manager several times a day, I think that the security should not be sacrificed for ease of use.

## Common Password Requirements

To understand the type of passwords the program would have to be able to generate, I looked at the password requirements of a few websites that my client would use the password generator to create password for.

### Google



Google has a very unrestrictive password policy where they don’t require you to have numbers or special characters present in the password.

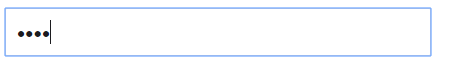
### Amazon



Amazon has slightly stricter requirements where they need numbers or special characters.

### Spotify





Spotify only requires any four characters for a password. In the screenshots above I used ‘aaa’ as the password that was deem too short whilst ‘aaaa’ is an accepted password.

Some sites do not allow for consecutive characters e.g. ‘12345’ or ‘abcde’.

From this I found a set of requirements that my password generator must be able to accommodate for.

# Documented design

## Overall Design

The application consists of three modules:

* Password manager (includes a login system)
* Password generator
* Password tester

For the most part, these modules are separate, however there are occasions where data will be passed between them, such as when a password is generated it can then be stored directly into the password manager. Also, if a password is deemed to be weak using the password tester, the user is directed to the password generator.

When the program is opened, the user is shown the main menu. Here there are options to use the manager, generator or tester.

If the user choses the manager, they can create a new user where they will enter a username and password, which will then be encrypted and hashed as needed and added to the database.

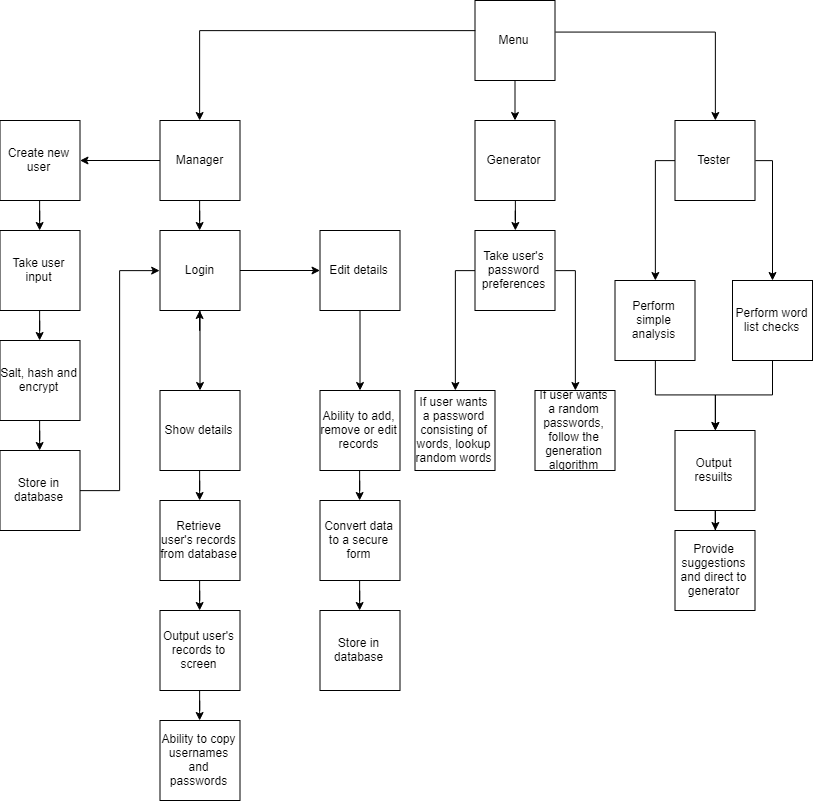
If the user already has an account, they will login which will query the database and check if the password, when hashed, matches the password hash stored for the username. If it does not, an error is shown.

Once logged into the manager, the user will see their stored passwords and be able to add, edit and remove records. Only the encrypted passwords will be initially visible.

The password generator will take a user’s preferences and generate either a word based or random string password which is then output to the screen.

The password tester will take the user’s password and perform a wordlist check and a basic format test. If the password is found in a wordlist, or it does not meet the basic requirements, the user will be directed to the password generator.

The operation of the application is shown below:



## Modular Design

* Main Menu (form 1)
  + Manager Login/Create user (form 2)
    - Login
      * Once logged in show user’s stored details (form 2.1)
      * Add/remove/edit entries
        + Update database
    - Create user (form 2.2)
      * Take input
      * Encrypt username, salt and hash password then store in database
      * Direct to login
  + Generator (form 3)
    - Take user’s preferences
    - Generate passwords based on user’s preferences
    - Output to screen
    - Go to form 1
  + Tester (form 4)
    - Take user’s password and choice
    - Perform chosen analysis. Either simple analysis, work list checks or both
    - Output results
      * Provide suggestion based on results
      * Direct to password generator
      * Go to form 1
  + Exit

## Data storage

### File structure

All files will be held inside a single folder. This will contain the executable, the database and the text files required for the password generation and testing.

### Entity Relationship Diagram

Two separate database tables are required to store the login details and the user’s usernames and password.

LoginInformation

Manager

### Entity Descriptions

LoginInformation (UserID Username, Password, Salt)

Manager (EntryID, Username, Password)

## SQL

### Create tables



### Login/create new user

/\* Create new user \*/

/\* Check if username is taken \*/

SELECT UserID FROM LoginInformation WHERE username = ?

/\* Add username and hashed password to users table \*/

INSERT INTO LoginInformation(username, password) VALUES (?, ?)

/\* Login \*/

/\* Get the ID, username and hashed password associated with a username \*/

SELECT UserID, username, password FROM LoginInformation WHERE username = ?

### Get passwords from manager associated with userID

SELECT password FROM Manager.password WHERE userID = ? VALUES (?)

## Data volumes

The executable will be one more than a few megabytes once it is compiled without debug information. The two text files will combine to around 1.5 megabytes and the database will likely be no more than 200 kilobytes since I expect there will be less than five overall users who will each store no more than 10 passwords with the average being 6.5 as shown by the study referenced above.

## Algorithms

### Encryption

For encryption I will be using the RC4 algorithm which is a stream cipher. This algorithm was once widely used including being used in WEP. It fell out of favour after multiple ways of attacking it were found. However, these attacks take multiple days and trained individuals to execute which is highly unlikely to happen in the client’s case. The way to defend against most of these attacks is to discard the keystream once it is used and ensure keys are random and not related.

The algorithm is made up of two main functions: the key-scheduling algorithm and pseudo-random generation algorithm. The outputs of which are then XOR’d together to create the cipher text.

#### Key scheduling algorithm

**for** i **from** 0 **to** 255

S[i] := i

**endfor**

j := 0

**for** i **from** 0 **to** 255

j := (j + S[i] + key[i [mod](https://en.wikipedia.org/wiki/Modulo_operation) keylength]) mod 256

swap values of S[i] and S[j]

**endfor**

#### Pseudo-random generation algorithm

i := 0

j := 0

**while** GeneratingOutput:

i := (i + 1) mod 256

j := (j + S[i]) mod 256

[swap values](https://en.wikipedia.org/wiki/Swap_(computer_science)) of S[i] and S[j]

K := S[(S[i] + S[j]) mod 256]

output K

**endwhile**

The pseudo-random generation algorithm produces a 255-byte long keystream based on the key scheduling algorithm which is then XOR with the plaintext to form the cipher text. To get the plaintext from the cipher text you simple XOR the cipher text with the keystream.

To protect against other common attacks on stream-ciphers such as the reused key attack, the key must have a variable component. This is recommended to be around 24 bits therefore I will append three characters to the end of the string. Since it is highly unlikely that the users of the manager store more than 263 passwords there will be a low chance that the keys repeat and can then be subject to a reused key attack.

#### Encrypt



#### Decrypt



### Hashing

This function utilises binary shift, XOR and AND to produce a hash from a string. It produces a large integer value which can then be converted to hexadecimal to produce a fixed length string which will then be stored in the database in the LoginInformation table.



### Login

The password hash associated with the username entered will be fetched from the database by an SQL query. The password that the user entered will then be hashed and matched against the hash from the database. If they are the same, the login is successful.



### Password generation

#### Random generation

The user will have choice over the following options:

* Length (minimum 8)
* Include capital letter
* Include number
* Include a special character

After the options have been chosen, a string of random lowercase letters (a..z) is made. Then up to two characters in the string will be replaced per option selected by the user. When the user selects an option, a variable holding the state of the option will be set to true. Before numOfCaps etc. is set, there must be a check to see if the user selected that option. If they have not selected the option, numOfCaps etc. it will remain at 0. When the user choses to add a random number of options the process of replacing a random number of character with a randomly generated character makes it extremely difficult to predict the output of the generator.

Following is a pseudocode algorithm assuming all options were chosen:



### Word based generation

The user specifies the number of words they want their password to consist of. For each word a random number is generated from 0 to the maximum number of lines in the dictionary. Then a loop moves to that line and appends it to a string which at the end of the outer loop will be the final password consisting of the number of words specified by the user. 

### Password Testing

#### Word list check

A text file will be opened and then searched for the password the user entered. If it is found, they will be directed to the password generator and given simple tips on how to improve their password.



The search function will be implemented as a binary search since the wordlist is large and will be sorted in alphabetical order. A linear search would be easier to implement but could take up to 1,000,000 (approximate size of list) comparisons to find the value, whilst the binary search will take at most 34 comparisons since it has a time complexity of O(logn)

##### Binary Search



Here the word list is loaded into an array. It is then searched using the binary search algorithm. Once the item is found, the Boolean ‘found’ will be set to true and the result will be returned to the user. This algorithm could also be implemented recursively

#### Simple password check

After analysing the password requirements of multiple services, including ones frequented by the client, I found that the following requirements would cover almost all common password requirement schemes: minimum eight characters, at least one uppercase letter, one lowercase letter, one number and one special character. Therefore, the simple password check should accept or reject passwords based on if they pass at least four out of five of these requirements.

The length of the password will be determined using a built-in function. To verify whether a password meet the other requirements, a series of regular expressions will be used. Each time a test is passed, the counter is incremented. If the counter is greater than or equal to four at the end of the tests, the password is allowed. The removal of the requirement to pass all tests allows the leeway needed for the tester to be as versatile as possible. If the count is less than four, the areas to improve the password will be displayed.

The pseudocode for determining whether a password meets the requirements for length and number of digits is show below. The process is similar for all other requirements.



# Testing

## Test stratergy

I used a bottom up stratery to implement the application where I created an array of console programs that implemented group of objectives. This allowed me to thoroughly test the functions in isolation before testing them again whilst compiling the application.

## Test data set

### Manager

#### Create user

When creating a user, any username can be used in combination with any password above seven characters. Therefore, a list of usernames of varying length and character combinations will be required. This will test what input the manager can take.

The list of passwords required must be of varying length, starting at less than seven and going up to around 100 characters. This will test the hashing algorithm and if the database can store a large input.

#### Login

At login the user enters and username and password which is then checked against credentials held in the database. Therefore, the data set must contain credentials currently held in the database and ones that have not been added.

#### Adding records to manager

### Tester

asd

### Generator

asd

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Data | Module | Type | Expected Result | Actual Result | Pass/Fail | Reference | Note |
| 1 |  | Random password generator |  |  |  |  |  |  |
| 2 |  | Passphrase generator |  |  |  |  |  |  |
| 3 |  | Simple password test |  |  |  |  |  |  |
| 4 |  | Word list test |  |  |  |  |  |  |
| 5 |  | Login to manager |  |  |  |  |  |  |
| 6 |  | Create user in manager |  |  |  |  |  |  |
| 7 |  | Connect to database |  |  |  |  |  |  |
| 8 |  | Show user’s records |  |  |  |  |  |  |
| 9 |  | User can add records |  |  |  |  |  |  |
| 10 |  | User can delete records |  |  |  |  |  |  |
| 11 |  | User can edit records |  |  |  |  |  |  |
| 12 |  | Can encrypt text |  |  |  |  |  |  |
| 13 |  | Can decrypt text |  |  |  |  |  |  |
| 14 |  | A user’s cipher text cannot be decrypted by another user |  |  |  |  |  |  |