EECS 448 Final Project

**Encrypted Account and Password Management Tool**

Gehrig Keane: 2727430

Kurtis Slagle:2742806

Philip Wood:*fill in KUID*

**Updated Proposal Items**

* The user class mentioned in the proposal was phased out with the implementation of the database. It isn’t necessary to maintain a class based on database information when the database manages entries and the like for us. Reading in database entries and storing them in an object would increase overhead and areas for gaps in security.
* We’ve selected SQLite as the database methodology for this assignment, Qt (the graphics framework we are using has a SQLite API for database interaction without network support. SQLite was selected due to its friendly interaction with Qt and its simple easy to use command line functionality.
* Qt was the selected GUI implementation which uses C++ source. Python was not used.
* The projects has not been packaged into a closed source executable due to licensing issues.

**Discussions on GUI design**

*Phillip said he’d do this part <LOL>*

**User Stories and Tasks**

A user’s day to day life requires recalling multiple account names with unique passwords in a timely fashion. Often menial accounts for services that aren’t commonly used must be recorded.

User Story 1:

* As a user looking to manage his/her account-password pairs, I’d like to log in to the management software securely. Authentication of the login credentials must be encrypted and queried with the software’s database.

Task 1:

* The login dialog associated with logging into the account management software must authenticate the login credentials with a universal encryption scheme (RSA encryption). This requires encrypted database queries in the authentication function.

User Story 2:

* As a user new to the account management software, I’d like to be able to register at the login prompt to start managing my own account-password pairs. Registering a new account with the software requires querying the database in a secure encrypted fashion, to augment the user base.

Task 2:

* Registering an account with the account management software requires a new dialog with a username field and two password fields for password confirmation. Upon acceptance any invalid proposed username and password combinations would result in an instruction dialog clearly informing the user of their error.

User Story 3:

* After I’ve logged in to the account management software, I’d like to be able to add/delete/edit account-password pairs. Addition/Deletion/Edition of a user’s data must be handled securely and all information in the database must be encrypted for the sake of privacy.

Task 3:

* Upon logging into the account management software the account-password pairs must be displayed to the user in a clean fashion such that the user can select different accounts and edit them appropriately. This may include adding a new account, deleting old accounts, or updating the information for current accounts. On the backend encrypted queries communicated with the database.

User Story 4:

* As a user who’s logged into the account management software, I’d like to be able to logout of the software.

Task 4:

* The user must be able to logout of the account management software and efficiently save the modification to their account.

**Testing Processes and Traceability Matrix**

* Security:
  + Inputs: User Credentials
    - Test for validity
      * Invalid credentials do not gain access
      * Users that already exist cannot be re-registered
    - Users may not enter empty strings
    - Warn users if credentials are invalid or the entered string is invalid
  + Users may not view any other users’ data
  + Data cannot be viewed outside program without decryption key
    - Viewing entries with a database viewer will show encrypted data
  + Any data sent to the database must fully encrypted
  + Any data coming out of the database must be decrypted only when being shown, but be stored in RAM in an encrypted state
  + Change encryption key if necessary
* Stability:
  + Arbitrary number of log-ins per session
    - Should be able to log in and out as many times as necessary for any number of users
  + Arbitrary number of database additions/deletions per session
    - Should be able to add/delete any number of accounts to/from the database
  + User may enter unexpected strings into fields, including spaces and empty strings
* Accuracy
  + The database must persist indefinitely
    - Test database after multiple changes to test for persistence
  + The changes must be immediate and immediately viewable by the user
  + There must be zero loss of user data

**Requirements Traceability Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RTM | Defect | Critical Failure | Repaired Failure |  |  |
|  |  |  |  |  |  |
| Test Case | Scenario | Steps | Test Data | Result | Defects |
| 1 | Log In | 1) Open Application 2) Enter Login Credentials | id = kurt pw = kurt | Pass |  |
| 2 | Add Account to Database For Existing Users | 1) Enter Account Details 2) Click "Add Account To Database" | id = hello2 pwd = 123123 | Pass |  |
| 3 | Maintain New Account Data after logout | 1) Log out 2) Log In 3) Check for hello2 account |  | Pass |  |
| 4 | Check Account Persistance Between Application Launches | 1) Log In 2) Add Account 3) Log Out 4) Log In 5) Check for New Account | id = hello2 pwd = 123123 | Pass |  |
| 5 | Register New User and Add Account Info |  |  |  |  |
| 5a | Register New User | 1) Register New User | id = new pwd = new | Pass |  |
| 5b | Add Account Info For New User | 1) Log in as new user without  any into in database 2) Add account info | id = 444 pwd = 444 | Pass |  |
| 6 | Register Multiple Users In One Sessions or after Log in | 1) Register New User 2) Log In 3) Log Out 4) Register New User | id = Hello pwd = Hello | Failure | Crash\_01: Program crash upon second account creation <New Account is successfully created and saved> |
| 6 | Register Multiple Users In One Sessions or after Log in | 1) Register New User 2) Log In 3) Log Out 4) Register New User | id = Hello pwd = Hello | Pass | Crash\_01 Repaired |
| 7 | Attempt to register existing User | 1) Click Register 2) Enter credentials for existing user 3) Expect register failure |  | Pass |  |
| 8 | Attempt to read database without decryption | 1) Access database with a database viewer without decrypting data | Expect incomprehensible Information | Pass |  |
| 9 | Delete accounts from database | 1) Log in 2) Delete existing account 3) Log Out 4) Log back in 5) Ensure account is deleted | <delete any existing account info> | Pass |  |

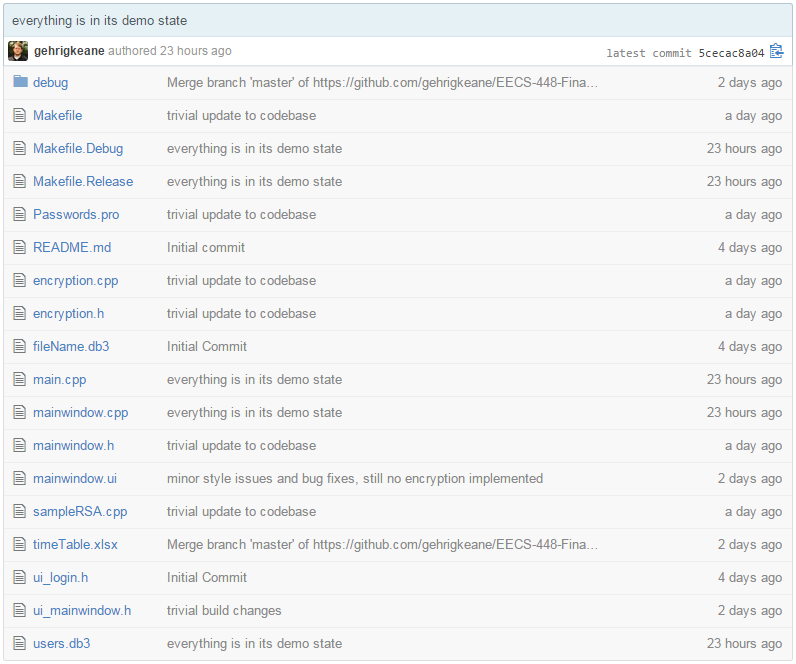
**Accuracy, efficiency, friendliness, and visual communications**

The implementation of the account management suite designed in our final project provides accuracy, efficiency, friendliness, and effective visual communication in the following ways:

* Accuracy: the major point of interest for accurate calculation within our software falls on the encryption scheme. Our software accurately takes standard strings encrypts and stores, then decrypts and displays said strings without loss of the user’s information.
* Efficiency: the efficiency associated with our permanent storage solution and the queries associated with said permanent storage are the most pertinent to discussions of efficiency. Our database is held in a db3 file which is managed by SQLite queries, meaning queries on a largely populated data set of users and account-password pairs will never take longer than tens of milliseconds even when queries are performed in real-time per the user’s direction. With a data set the size of our current database file queries are generally less than one millisecond.
* Friendliness: We’ve designed a dialog class (the invalidDialog function) that displays a simple dialog with a text message and an ok button. We feel this often instructional dialog helps users that aren’t familiar with the software gain insight into their mistakes or lack of understanding when working with their account. These instructional dialogs inform the user if their passwords don’t match when registering, it also informs the user if the entered login username isn’t found in the database, et cetera. Furthermore, we believe the UI in the MainWindow class clearly and in a simple fashion displays all the information associated with a user’s account.
* Visual Communication:

**GitHub Usage**

*https://github.com/gehrigkeane/EECS-448-Final-Project*

There were a total of 22 commits and no major forks for the sake of simplicity. 

Concluding Remarks and Future Extensions

References