The SEcube Wallet

Project documentation

Release: April 3rd, 2019

Proprietary Notice

Authors

Giuseppe AIRÒ FARULLA *(CINI Cybersecurity National Lab)* giuseppe.airofarulla@polito.it

Walter GALLEGO GÓMEZ *(CINI Cybersecurity National Lab)* waltergallegog@gmail.com

Paolo PRINETTO *(President, CINI)* paolo.prinetto@polito.it

Antonio VARRIALE *(Managing Director, Blu5 Labs Ltd)* av@blu5labs.eu

Trademarks

Disclaimer

Table of content

1. Introduction

Nowadays, having a large quantity of digital passwords is the norm, and as their number increases, it becomes impossible to memorize all of them. This is especially true considering one should adopt strong passwords which are in general very long and complex, and therefore hard to remember. This has led users to rely on software applications to manage their passwords, the most common cases being web browsers and password wallets. The drawback of this approach is that security may be compromised, since all the passwords are stored in the same place and an attacker could gain access to them. For users particularly interested in ensuring the security of their systems, this software-based approach may not be acceptable.

This document presents **SE***cube***Wallet**, a custom solution based on the **SE***cube*™ framework to securely manage passwords and credentials in general. The **SE***cube*™ (Secure Environment cube) *Open Security Platform* is an open source security-oriented hardware and software platform, designed and constructed with ease of integration and holistic security in mind.

The **SE***cube*™ (Secure Environment cube) framework consist of an open source security-oriented hardware platform designed by the Blu5 Group, and a set of open source software libraries developed by European research institutions. The core of the framework is the **SE***cube*™ chip, which integrates three key security elements in a single package: a fast floating-point Cortex-M4 CPU, a high-performance FPGA and an EAL5+ certified Security Controller (Smart Card). These elements, in conjunction with a set of custom software libraries allow developers to implement highly reliable security applications.

2. The SE*cube*Wallet

2.1. Goals

The main goals of the **SE*cube*Wallet** are:

- Managing the users’ passwords (hereinafter referred to as a *wallet*) using secureSQLite, one of the **SE*cube***™ host-side libraries, that works by wrapping the functionalities of the SQlite standard to create **SE*cube***™ secured databases (cfr. “Wiki”).
- Serving as a general GUI to let users authenticating to the **SE*cube***™ device and creating, opening, editing, saving, and deleting wallets with ease. The GUI displays the wallet's content in a table view and each column can be filtered individually.
- Provide the users with strong passwords (and passphrases) which can be used with confidence in any login service. The application also verifies the entropy (strength measure) of both the passwords it generates itself and the ones provided by the user.

2.2. Interaction with the SE*cube*™ framework

The **SE*cube*Wallet** interacts with the **SE*cube***™ device, requesting for its basic services like authentication and encryption.

The application runs in a computer, denominated *host*, and the **SE*cube***™ chip, denominated *device*, has to be connected to it via USB. The host request for security operations to the device, using the **SE*cube***™ software libraries. Each device comes with a master password, and in order to use it, it is necessary to perform an authentication procedure entering said password from the host.

It manages passwords using secureSQlite, one of the **SE*cube***™ libraries, which wraps the functionalities of the SQlite standard to create **SE*cube***™ secured databases. In short, the data of interest is encrypted using the **SE*cube***™ device, and can only be decrypted if the device is connected and the user authenticates using a master password.

As the core operations are performed by the device, not by the host, the encryption/decryption can be done in any computer where an appropriate version of Qt is installed and the device is connected.

2.3. Technologies and frameworks

The desktop application presented in this document is written in C/C++ and Qt. It can be modified using the Eclipse IDE[1] release Neon version 4.6.3 (or higher) or the Qt Creator IDE[2] version 4.6.2 (or higher). Additionally, the application uses the Open Source freely available libraries PwGen[3] (random password generator) and zxcvbn[4] (password strength estimator), together with a custom passphrase generator.

[1] https://www.eclipse.org

[2] https://www.qt.io/qt-features-libraries-apis-tools-and-ide/

[3] https://linux.die.net/man/1/pwgen

[4] https://github.com/dropbox/zxcvbn

The development was carried out on a Linux-based machine, but the application has been tested with success also in machines running Windows or MacOS.

As front end, the application presents the user with a pleasant and intuitive graphical user interface. With it, the user can easily create, delete, open, and modify password wallets. The GUI is easily configurable and is cross-platform.

Additionally, the application can suggest strong passwords and passphrases and can also verify the entropy of the ones provided by the user.

2.3.1. Qt

SE*cube*Wallet was developed using the Qt framework, version 5.11.1[5]. Qt has been chosen for a variety of reasons, among which there are

- It allows for a seamless use of C/C++ libraries such as **SEFile** and secureSQLite, which are the backbone of this project.
- Qt is cross-platform, meaning the developed application can be compiled to work on any of the major OS.

Qt 5.10 or higher is required to compile **SE*cube*Wallet** from sources.

Qt is a cross-platform application development framework for desktop, embedded and mobile. Supported Platforms include Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and others. Qt is not a programming language on its own. It is a framework written in C++. A preprocessor, the MOC (Meta-Object Compiler), is used to extend the C++ language with features like signals and slots. Before the compilation step, the MOC parses the source files written in Qt-extended C++ and generates standard compliant C++ sources from them.

2.3.2. PwGen: Pronounceable password generator

PwGen is an open source program that generates human friendly passwords that are also secure. It is available in the official Linux repositories, and there is a Windows version as well.

PwGen offers several options that can drastically change the nature of the generated passwords. Here follows a list of the options available for users of the **SE*cube*Wallet**:

- Length: The desired length of the password. It is recommended to be at least 12 for non-random passwords and 8 for random ones.
- No numerals (-0): Don't include numbers in the generated passwords.
- No capitalize (-A): Don't include any capital letters in the generated passwords.
- Ambiguous (-B): Don't use characters that could be confused by the user when printed, such as 'l' and '1', or '0' or 'O'. This reduces the number of possible passwords significantly, and as such reduces the quality of the passwords. It may be useful for users who have bad vision, but in general use of this option is not recommended.
- Capitalize (-c): Include at least one capital letter in the password.
- Numerals (-n): Include at least one number in the password.
- Secure (-s): Generate completely random, hard-to-memorize passwords.
- No vowels (-v): Generate random passwords that do not contain vowels or numbers that might be mistaken for vowels. It provides less secure passwords to allow system administrators to not have to worry with random passwords accidentally contain offensive substrings.

[5] https://www.qt.io

- Symbols (-y): Include at least one special character in the password.

By default, pwgen behaves as if the options **-nc** were used, that is, pronounceable passwords with at least 1 capital letter and 1 number. The strongest passwords this program can generate are obtained with the options **-ys**, as it results in random passwords with special symbols, numbers and capital letters (these last two are enabled by default). They are very hard to remember, but can be managed easily via the **SE*cube*Wallet** application.

2.3.3. zxcvbn: Password strength estimation

For over 30 years, password requirements and feedback have largely remained a product of LUDS: counts of Lower- and Uppercase letters, Digits and Symbols. LUDS remains ubiquitous despite being a conclusively burdensome and ineffective security practice. zxcvbn is an alternative password strength estimator that is small, fast, and crucially no harder than LUDS to adopt.
zxcvbn is regarded by the community as one of the most reliable and mathematically advanced open source password estimators. In security forums and discussion, it always pops out as an excellent tool, much better than other passwords estimators commonly used in web pages.

zxcvbn calculates a password’s entropy to be the sum of its constituent patterns. Any gaps between matched patterns are treated as brute-force "patterns" that also contribute to the total entropy. That a password’s entropy is the sum of its parts is a big assumption. However, it’s a conservative assumption. By disregarding the "configuration entropy" — the entropy from the number and arrangement of the pieces — zxcvbn is purposely underestimating, by giving a password’s structure away for free: It assumes attackers already know the structure (for example, surname-bruteforce-keypad), and from there, it calculates how many guesses they’d need to iterate through. Currently zxcvbn matches against:

- Dictionaries: Common words the user is likely to use as password. Multiple dictionaries, in a simple .txt format can be used. In this work, we present a few: English words, Italian words, names and surnames, Burnett’s 10,000 common passwords, words from tv and films. The match has an associated frequency rank, where words like 'the' and 'good' have low rank, and words like 'photojournalist' and 'maelstrom' have high one. This lets zxcvbn scale the calculation to an appropriate dictionary size on the fly, because if a password contains only common words, a cracker can succeed with a smaller dictionary. For all dictionaries, match recognizes uppercasing and common substitutions.
- Spatial keyboard patterns: Some users are likely to choose passwords based on spatial pattern. Qwerty keyboard, Dvorak keyboard, and keypad are considered.
- Repeats: Users are also prone to repeat the same characters in a password.
- Sequences: Numeric or alphabetic sequences.
- Dates: Years from 1900 to 2019 are considered and dates in different formats (e.g., 3-13-1997, 13.3.1997, 1331997).

2.3.4. Custom passhprase generator

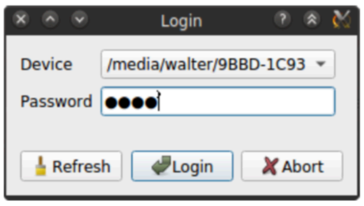
PwGen can generate pseudo-random pronounceable passwords, but to match the constraints proposed by the zxcvbn library they should be easy to remember, but long enough to give excellent entropy results. To fill this gap, a custom PassPhrase generator was developed.

The PassPhrase generator developed by the author works by randomly picking out words from dictionary files. The user can tune the PassPhrase generation as follows:

- **Dictionaries**: The user must select appropriate dictionaries, containing a sufficiently large number of lines (larger than 10000) to ensure the picked words are really random. The English and Italian dictionaries used by zxcvbn are a good example. The user can work with as many dictionaries as desired, and the format must be one word per line. Only the first word of each line is counted, as everything after a space is trimmed.
- **Number of words**: The user can configure the number of words the generated PassPhrases are composed of. The recommended size is four, but it can be as long as the user wants.
- **Minimum Length of Words**: With this option is possible to select only random words whose length is higher than a certain value. This is to make sure the resulting PassPhrase is not too short and therefore too insecure. The drawback here is that the higher the selected threshold, the fewer the available words in the dictionaries.
- **Only use infrequent words**: If the dictionaries follow the same format as those used for zxcvbn, that is, the words are ordered by frequency, having the most uncommon words in the lower part of the dictionary, the user can then ask to generate PassPhrases containing only unusual words. The drawback here is, again, fewer words to choose from. The percentage of words that are used is configurable.
- **Capitalize first letter**: To make the PassPhrases more readable, the first letter of each word can be capitalized.

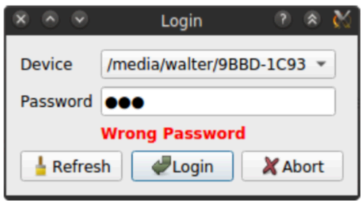
2.4. Using the Se*cube*Wallet

In order to compile the project with Qt (version 5.10 or higher required), open Qt and open the project by selecting the file “Secure Wallet/code/secubewallet-master/SecubeWallet.pro”. Once the project is imported in order to start building the user need to first build each “sub-modules”. Expand the project directory in the left-menu, than starting from the SEfile directory, right-click on it and select “compile”. Proceed to build each “sub module” with a bottom up approach, once each module has been correctly compiled the user can compile the whole project. From now is not required to build each “sub module” every time unless a clean is performed.



Device /media/walter/9BBD-1C93
Password ●●●●

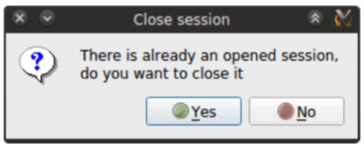
(a) Login Dialogue



Device /media/walter/9BBD-1C93
Password ●●●

Wrong Password

(b) Wrong Password



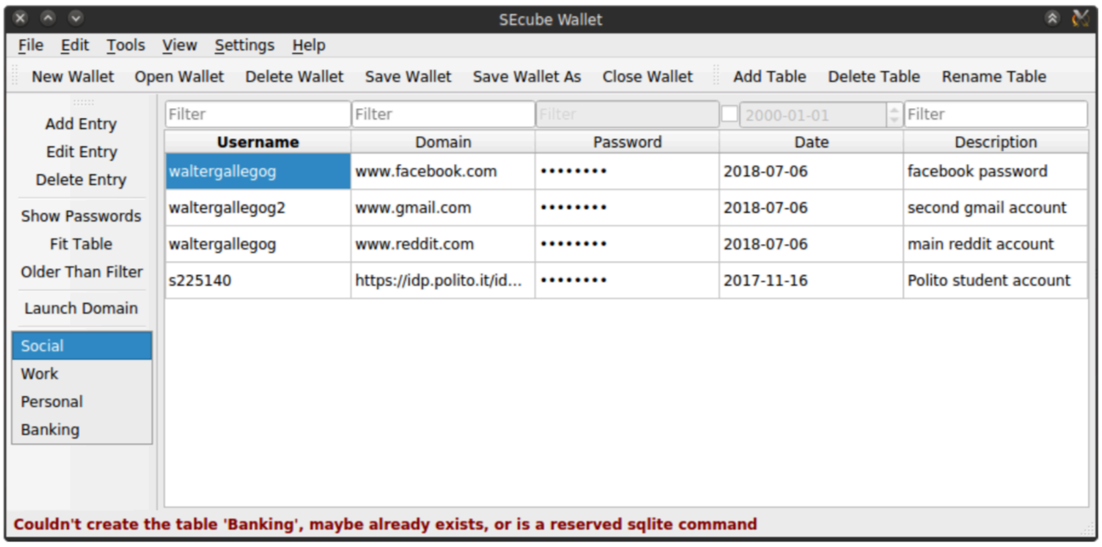
There is already an opened session, do you want to close it

(c) Already an open session

Figure 1 - Login windows

The main window is composed of the following elements:

- **Table View:** Used for displaying the wallet entries. It resizes smoothly with the window, can be ordered by any of the columns, and the pass- words are hidden by default but can be shown if the user wants to.
- **Filters:** The user can search in each of the table's columns using filters. These filters are implemented inside a separate container, but they resize together with the table.
- **Entries Tool Bar:** It is positioned to the left of the table. It has the actions: add/edit/delete entries, show passwords, fit table, change date filter, launch domain and select table.
- **Tables Tool Bar:** It is positioned in the top right of the table. It has the actions: add/rename/delete table.
- **Wallets Tool Bar:** It is positioned to the top left of the table. It has the actions: new/open/delete/save/save as/close Wallet.
- **Menu Bar:** It is positioned at the top of the window. It contains all the previous actions, plus preferences and help.
- **Status Bar:** Positioned at the bottom of the window, it is used to display some success/error messages to the user, and the current wallet name.



Username	Domain	Password	Date	Description
waltergallegog	www.facebook.com	••••••••	2018-07-06	facebook password
waltergallegog2	www.gmail.com	••••••••	2018-07-06	second gmail account
waltergallegog	www.reddit.com	••••••••	2018-07-06	main reddit account
s225140	https://idp.polito.it/id...	••••••••	2017-11-16	Polito student account

Couldn't create the table 'Banking', maybe already exists, or is a reserved sqlite command

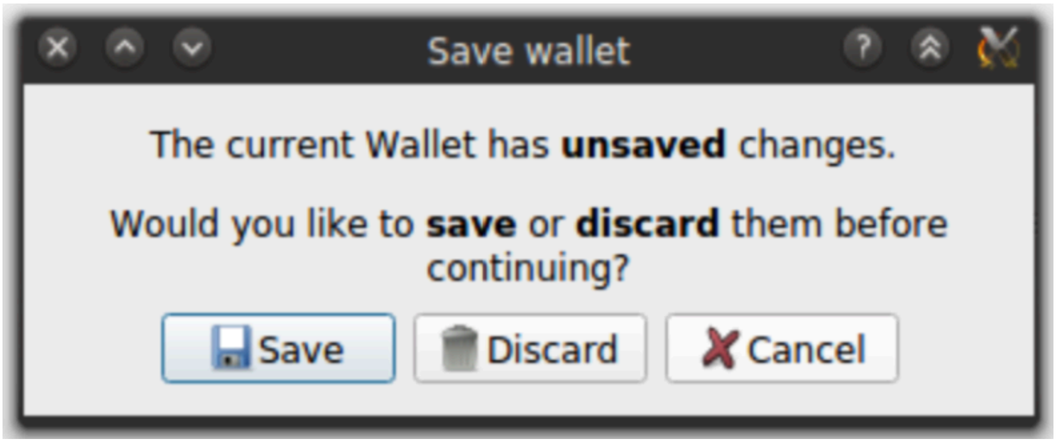
Figure 2 - Main window

2.4.1. Managing wallets in the *Secube*Wallet

The actions to manage users’ wallets are: New, Save, Save As, Open, Close, and Delete.

New Wallet action

When the user triggers the **New_Wallet** action, the first step to execute is to check if there is another wallet opened and if it has unsaved changes. If, so the confirmation dialogue in Figure 3 is shown, so the user can decide whether to save the changes, discard them, or cancel the creation of a new wallet.



Save wallet

The current Wallet has **unsaved** changes.

Would you like to **save** or **discard** them before continuing?

Figure 3 - Save or Discard confirmation window

In case the user clicks Save, the Save_Wallet action is triggered before continuing. Discard continues without saving, and Cancel returns without doing anything.

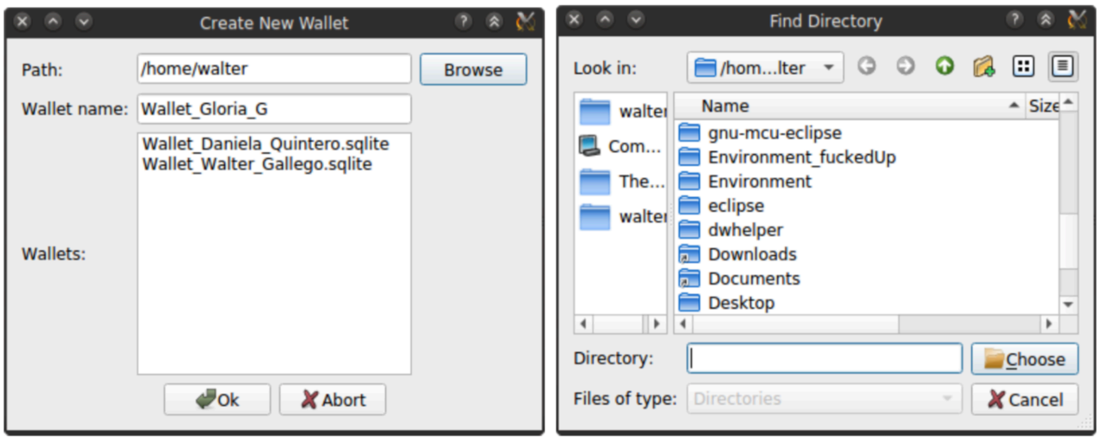
If the process continues, the next step is to close any previous in-memory database handlers, save the table_view geometry (if any), and open a new in-memory database using the Qt class QSqlDatabase.

An in-memory data base is used for editing. It has the advantage of being fast because there is no access to the hard disk, and secure, because all the data is in the application memory space, and therefore is protected by the OS.

The last step is to update the GUI state, by enabling some action like Add_Table and Save_Wallet, and disabling others, like Delete_Table and Rename_Table.

Save Wallet action

To write the wallet to the disk, it is necessary to have a filename, so the first step is to check if the user already entered one (from previous saves). If not, with the dialogue windows shown in Figure 4, the user can choose the directory and the filename to save.



Path: /home/walter
Wallet name: Wallet_Gloria_G
Wallets: Wallet_Daniela_Quintero.sqlite
Wallet_Walter_Gallego.sqlite

Look in: /hom...lter
walter
Com...
The...
walter
Name | Size
gnu-mcu-eclipse
Environment_fuckedUp
Environment
eclipse
dwhelper
Downloads
Documents
Desktop
Directory:
Files of type: Directories

(a) User can enter a New name. Current wallets are displayed
(b) if `Browse` is clicked a `QFileDialog` is launched

Figure 4 - Window for saving wallets and managing file names

The need for two dialogues instead of a regular file browser comes from the fact that the chosen filename will not be readable from the OS, since SEFile also encrypts it. Similarly, wallets already saved in the directory cannot be displayed with a regular file browser, so it is necessary to use the SEFile function secure_ls.

After having chosen a filename, the next step is to read all of the tables in the current in-memory database, row by row. Then, for each row to save, the **Save_Wallet** action creates and stores in a temporary file one SQLite statement of the form *INSERT INTO table VALUES(row)*. All of them are then merged into a single command statement that is then executed into the secured in-disk database. This optimization ensures only one access to the SE*cube*™ and hard disk is performed.

Save Wallet As action

This action is very simple, it just clears the current filename (if any), and calls the **Save_Wallet** action; as there is no filename, the user is forced to enter a new one. The only point to be careful about is that, in case the **Save_Wallet_As** process is aborted, the previous filename needs to be recovered, so before clearing, the filename is temporary stored in case it is needed.

Open Wallet action

Similarly to what happens with the **New_Wallet** action, also in case of the **Open_Wallet** action, the first step is to check for unsaved changes and in case to prompt the user whether they have to be saved or discarded, with the dialogue in Figure 3.

If the user decides to continue, the dialogues in Figure 5 allow to choose from the list of the existing wallets the one to open.

The application proceeds doing the inverse process of the **Save_Wallet** action, that is, read all the tables from the secure in-disk database and create an in- memory database with this data. First, the in-disk data base is opened as read only, and a list named tables with the existent tables in it is generated. Then an in-memory database is created. Finally, for each table in tables, its contents are read, a correspondent table is created in the in-memory database, and the latter is populated with the read contents from the in-disk DB.

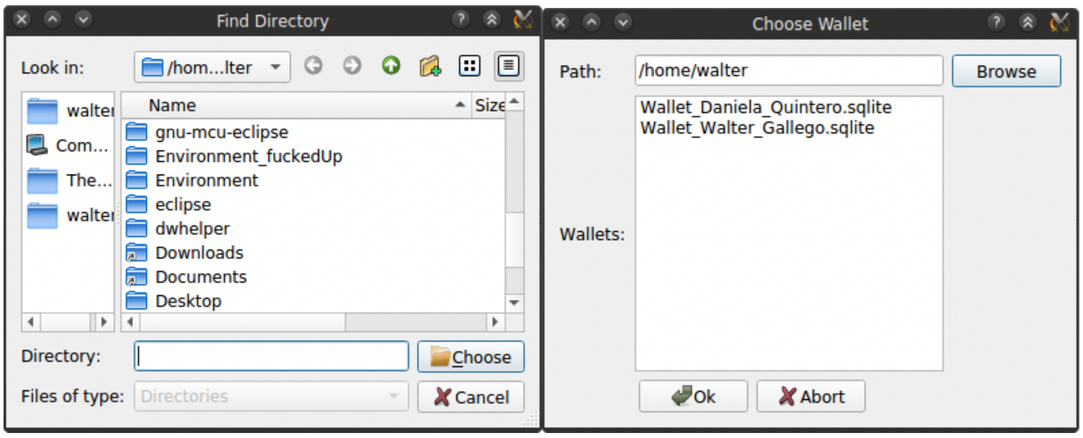


Figure 5 - Opening wallets action

Close Wallet action

This action is very simple. As usual, before closing the current wallet a check for unsaved changes is performed, and the user is asked what to do with them using a confirmation dialogue. If the user decides to continue, the in-memory database handler is closed, and the table geometries are saved. Finally, the GUI is updated.

Delete Wallet action

Deleting a wallet involves deleting an in-memory database and/or an in-disk database.

If only the in-memory database exists (i.e., the user has not saved it to disk yet), the wallet is simply closed, as in the previous section.

If there is no opened wallet, and the user wishes to delete an in-disk database, a select file dialogue equal to the one in the **Open_Wallet** action is shown, where the user can choose the wallet to delete. Then the wallet is deleted: the encrypted version of its filename is retrieved using the **SE***cube*™ API, and then the file in the disk is deleted using standard OS calls.

If both in-disk and in-memory wallets are to be deleted, that is, if there is an opened wallet that has been already saved to disk, there is no need for a select file dialogue, as the filename is known from the **Save_Wallet** action. The in-memory database handler is closed and the in-disk encrypted wallet file deleted as explained above.

In all the above cases, a confirmation dialogue asks the user for confirmation prior deleting the wallet.

2.4.2. Managing tables in the Secube*Wallet*

Actions regarding tables, their creation and display, are explained in the following. Qt fundamental input structures are used, namely:

- The QInputDialog class, which provides a simple convenience dialog to get a single value from the user;
- The QMessageBox class, which provides a modal dialog for informing the user or for asking the user a question and receiving an answer;
- The QComboBox class, which provides a widget for a combined button and popup list.

Add Table action

With an in-memory wallet opened, the user can add a new table to it by simply entering a name. With the QInputDialog class, it is possible to ask the user for a name easily. If the name is valid, the SQLite query to add the table is executed. It may be the case the table already exists, in which case the query execution returns an error and the user is notified.

Delete Table action

To delete a table, it is enough to use the SQLite DROP TABLE command. With the QMessageBox the user is asked to confirm before deleting. After deleting, the table view needs to be updated, to show the next table in the wallet. If there are no more tables, the view is simply hidden, and other GUI elements like Add Entry disabled.

Rename Table action

To Rename a table the SQLite command ALTER TABLE RENAME TO is used. After this, a QComboBox used to select the current table being displayed is updated to reflect the name change.

Select Table

To allow the user to select one table to display out of the existent ones in the current wallet, a QComboBox is added to the Entries Tool Bar. Each time a Wallet is opened/closed, or a table is added/renamed/deleted, the items in the QComboBox are updated accordingly. When the selected item in the QComboBox changes, the update procedure is triggered.

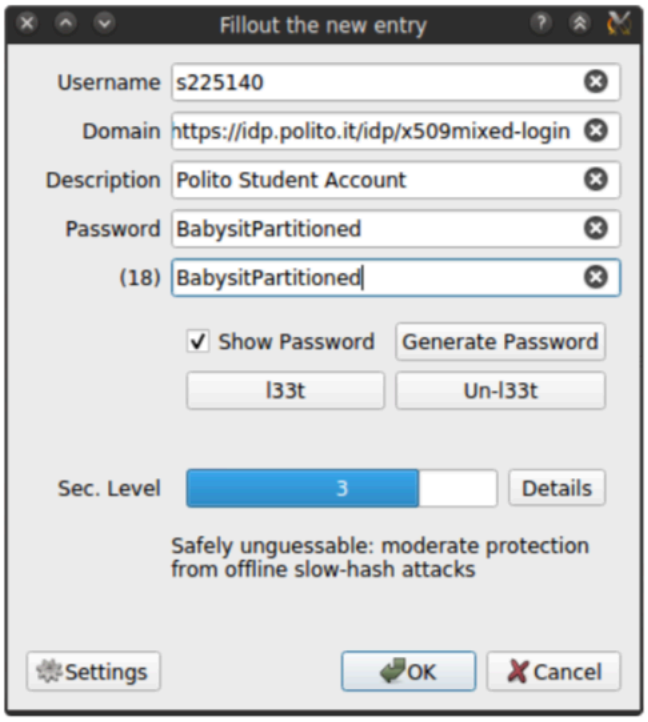
2.4.3. Managing entries in the Se*cube*Wallet

Actions regarding entries, their creation and display, are explained in the following

Add Entry action

The AddEntry class allows users to input a new entry to one of the tables, using the subwindow in Figure 6. This subwindow is composed of:

- **Text input elements:** To enter the Username, Domain, Description and Password, five QLineEdit are used. The password needs to be entered twice to make sure it is the desired one, and if it is not the same in both text fields, an error message is displayed. The date is not entered by the user, but generated automatically based on the system clock;
- **Show password checkBox:** By changing the echo mode in the pass- word QLineEdit, it can be hidden or shown. It is hidden by default;
- **Password Generator Button:** This triggers one of two available generators: PwGen or PassPhraseGen.
- **Password Strength elements:** A Progress Bar used to display the password strength calculated with the zxcvbn library, a label to show some information about the strength, and a button to open a subwindow showing the details of these calculations. If the zxcvbn library has not been compiled by the user in the settings window, these elements are disabled;
- **Bottom buttons:** The ok button is clickable only when all the text fields (except for description, which is optional) are filled, and the two passwords coincide. The settings button opens the settings subwindow so the user can customize the password generators or the strength estimator without having to close the AddEntry subwindow.



Username s225140
Domain https://idp.polito.it/idp/x509mixed-login
Description Polito Student Account
Password BabysitPartitioned
(18) BabysitPartitioned
✓ Show Password
Sec. Level 3
Safely unguessable: moderate protection from offline slow-hash attacks

Figure 6 – Window for adding a new entry to a table

When the user clicks the ok button, the new entry needs to be added to the database.

Edit Entry action

The user can edit any of the entries by selecting one of the cells in the tableView and clicking the Edit Entry button, or by double-clicking any of the cells. In either case, the data from the selected row is retrieved and passed to new AddEntry object using its constructor. In this way, the user is presented with an AddEntry subwindow where the input fields are already filled with the current data. The user can then modify and save them by clicking ok. In this process the ProxyModel is used instead of the model, because the former allows to identify the items selected in the tableView.

Delete Entry action

Deleting an entry is very simple. The row index of the selected cell is used in the ProxyModel method removeRow(row), and the change is submitted with submitAll. Before deleting, the user is asked to confirm the action.