SEcube™ Secure Wallet

Technical Documentation

Release: October 2021

Proprietary Notice

The present document offers information subject to the terms and conditions described hereinafter. The authors reserve the possibility to change the content and information described in this document and to update such information at any time, without notice. Despite the attention that has been taken in preparing this document, typographical errors, error or omissions may have occurred.

Authors

Matteo FORNERO *(Researcher, CINI Cybersecurity National Lab)* matteo.fornero@consorzio-cini.it
Nicoló MAUNERO *(PhD candidate, Politecnico di Torino)* nicolo.maunero@polito.it
Paolo PRINETTO *(Director, CINI Cybersecurity National Lab)* paolo.prinetto@polito.it
Gianluca ROASCIO *(PhD candidate, Politecnico di Torino)* gianluca.roascio@polito.it
Antonio VARRIALE *(Managing Director, Blu5 Labs Ltd)* av@blu5labs.eu

Trademarks

Disclaimer

Contents

1. Introduction

The **SE*cube*™** Secure Wallet (**SE*wallet*™**) is a password manager specifically developed for the **SE*cube*™** Security Platform[1]. A password manager is a software that helps users to create, store, and use passwords; it is not just an alternative to writing passwords on a notebook.
Password managers are designed to be secure, meaning that they help users to create strong passwords, they store them in secure devices (such as the **SE*cube*™**) and they implement additional measures to ensure a reasonable level of security (i.e., keeping track of data breaches leading to the leakage of usernames and passwords, forcing the user to update passwords periodically, etc.).
Password managers were not very popular in the past decades, mainly because normal people did not really have many passwords to remember. With the explosion of Internet and, in particular, with the invention of smartphones and e-commerce platforms, people usually have tens of passwords to remember. This is a very significant problem because, as well as the number of passwords of each user, also computing performance has increased significantly in the past decades. Many passwords that were strong enough in the early 2000's are not secure today, because traditional computers and tools can crack them easily. It is necessary, therefore, to use longer and more complex passwords; this is very difficult for the average user because complex passwords are almost impossible to remember. We need, therefore, tools that can help us remembering long and complex passwords: this is exactly the aim of password managers.

2. SE*cube*™ Secure Wallet

The main goals of the Secure Wallet (hereinafter referred to as '**SE*wallet*™** ') are:

- secure storage of passwords (a collection of passwords is a 'wallet') resorting to the capabilities of the **SE*cube*™** Security Platform;
- helping users generating strong passwords and passphrases, leveraging external tools and libraries to verify that each password meets specific requirements in terms of complexity (password entropy);
- helping users managing their passwords providing a graphical user interface supporting all the basic features that are needed by a password manager.

Describing the internal implementation of **SE*wallet*™** is out of the scope of this manual; however, it is important to give at least a high-level overview of the implementation.
SE*wallet*™ is based on the capabilities of the **SE*cube*™** Security Plaftorm, both in terms of hardware and software. The **SE*cube*™** is used to encrypt and decrypt the data managed by **SE*wallet*™** , using cryptographic keys that are permanently stored inside the **SE*cube*™** itself. All the information managed by **SE*wallet*™** is stored inside a SQL database[2], implemented resorting to the Secure SQLite Database Engine[3] that has been developed in conjunction with **SE*file*™** [4]. This results in an encrypted database file for each password wallet managed by the application; the encrypted database can be stored wherever the user wants, but the crucial aspect is that the keys to decrypt wallets are stored exclusively inside the **SE*cube*™** device and cannot be exposed outside of it.
The interaction between the GUI and the **SE*cube*™** consists of an intermediate software that communicates with the GUI through a TCP socket and acts as a dispatcher for the requests directed from the GUI to the **SE*cube*™** .
The overall scheme, therefore, is that the user only has to interact with a simple GUI; the graphical

[1] https://www.secube.blu5group.com/
[2] https://www.sqlite.org/index.html
[3] https://github.com/SEcube-Project/Secure-SQL-Database
[4] https://github.com/SEcube-Project/SEfile

interface provides all the features that are required to manage passwords and wallets. Internally, the GUI communicates through a socket with a dispatcher that manages the interaction with the **SE*cube*™** device.

3. System requirements

At the moment, **SE*wallet*™** is compatible with Linux operating systems; however, work is being done to make it compatible with Windows. In order to compile and use **SE*wallet*™** , you must satisfy the following list of requirements:

- Linux operating system (tested on Ubuntu 20.04 LTS, kernel 5.4.0-84-generic)
- GCC and G++ (tested version 9.3.0)
  To install GCC and G++, open a terminal and enter command:
  sudo apt-get install build-essential
- STM32CubeIDE (tested version 1.7.0)
  To install ST32cube IDE, visit the website:
  https://www.st.com/en/development-tools/stm32cubeide.html
- Qt framework (tested on Qt 5.14.2)
  To install Qt 5.x, open a terminal and enter the command:
  sudo apt-get install qt5-default
- Qt Creator (tested version 4.12.2)
  To install Qt Creator, open a terminal and enter the command:
  sudo apt-get install qtcreator

4. System setup

4.1. Preparing the SE*cube*™ device

For testing **SE*wallet*™** , it is recommended to use a **SE*cube*™** with a clean flash memory and the firmware from the SDK version 1.5.1 or 1.5.2. Follow the official wiki of the SDK (1.5.1 or 1.5.2)[5] to erase the content of the flash memory and program the firmware on the **SE*cube*™** . After flashing the firmware, keep following the wiki to initialize your device with a proper serial number and with the PIN codes for administrator mode and user mode. When you are done with the initialization, come back to this guide.

4.2. Compiling and launching the Secure Wallet application

These are the steps that you need to follow in order to compile and launch the application:

1. download **SE*wallet*™** from the GitHub repository[6] as zip file, or clone the repository;
2. if you downloaded **SE*wallet*™** as zip package, extract it wherever you want;
3. open STM32Cube IDE, you are free to choose the workspace that you prefer;
4. in STM32Cube IDE, select 'import existing project from filesystem' and browse to the folder where you cloned the repository (or extracted the archive);

[5] https://github.com/SEcube-Project/SEcube-SDK
[6] https://github.com/SEcube-Project/Secure-Wallet

5. inside the 'Source' folder, there is another directory named '**SE*wallet*™** Backend'. Select this folder for the import procedure, then go on with the guided steps for importing the **SE*wallet*™** Backend project;
6. compile the SEwalletBackend project in Release mode;
   The compilation in STM32Cube IDE must be done using the flag -std=c++17 for G++.
7. close ST32cubeIDE and open Qt Creator;
8. open the project SEwalletGUI.pro (the project file is located at /Source/SEwalletGUI/);
9. push *Configure Project* button;
10. set *Release* mode to compile the project (hammer);
11. run the project (green arrow).

Now you can start using **SE*wallet*™** . The program is very simple, we will describe its main features in the next section.

5. Secure Wallet GUI

5.1. Login

When the user starts the application, the login dialog is shown (figure 1). At the center of the window you have to select your **SE*cube*™** device and you have to enter your PIN code (user privilege level) to login on the **SE*cube*™** . In the top-right corner of the window there is a button to refresh the list of **SE*cube*™** devices attached to your computer, and another button to logout and exit from the application.

Figure 1: **SE*wallet*™** GUI - login window.

5.2. Creating a wallet for your passwords

Starting from the main window of the program, as shown in Figure 2, the user must click on the 'New Wallet' button in the top left corner of the screen. When a new wallet is created, it is always empty; the user can add new passwords by selecting the 'Add Entry' button. Similarly, passwords can be modified and deleted by means of dedicated buttons.
A wallet could be used to store a significant amount of passwords, so there is also the possibility of creating multiple tables for each wallet. If the wallet was a notebook, a table would be a page of the notebook; organizing the passwords of a wallet in multiple tables is useful to keep everything clean and tidy.
Notice that, when creating a wallet, the wallet is not automatically saved to the disk. This means that you have to save the wallet manually, clicking on the appropriate button, as soon as you want to synchronize the changes you made with the wallet file actually stored on the non volatile memory. This behaviour is due to the fact that, when a wallet is loaded, it is always managed as an 'in memory' database, meaning a database that is temporarily copied in RAM, so that interactions are faster. This is also required by the fact that the wallet file stored on disk is encrypted, so there is a significant computational overhead every time that the file on the disk is accessed, because of decryption.

Important notice: when a wallet is saved to disk, its content is encrypted. Encryption does include also the name of the file, that is replaced by the SHA-256 digest of the original name. If you browse the folder where you saved your wallet with a conventional file browser, you will not be able to see the real file name of your wallet. This is possible only using the **SE*wallet*™** application.

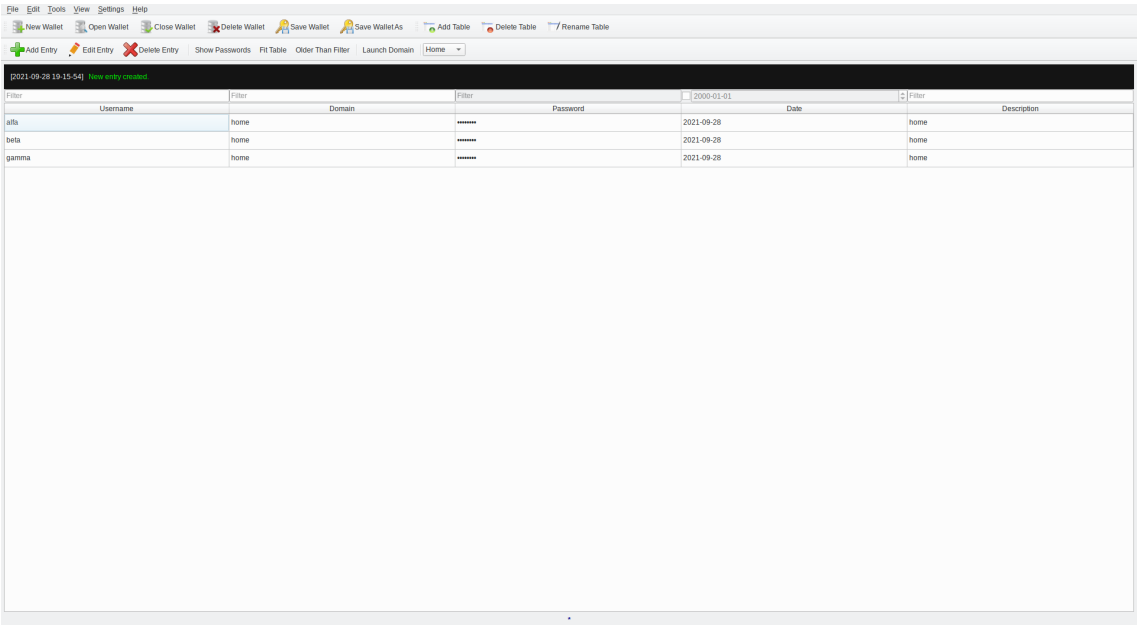

Figure 2: SEwalletGUI - main window.

5.3. Loading an existing password wallet

Loading an existing wallet is just as simple as creating a new wallet. Just click on the 'Open Wallet' button and browse to the folder where you stored your existing wallet (Figure 3), then open it. The wallet will be opened showing tables and passwords.

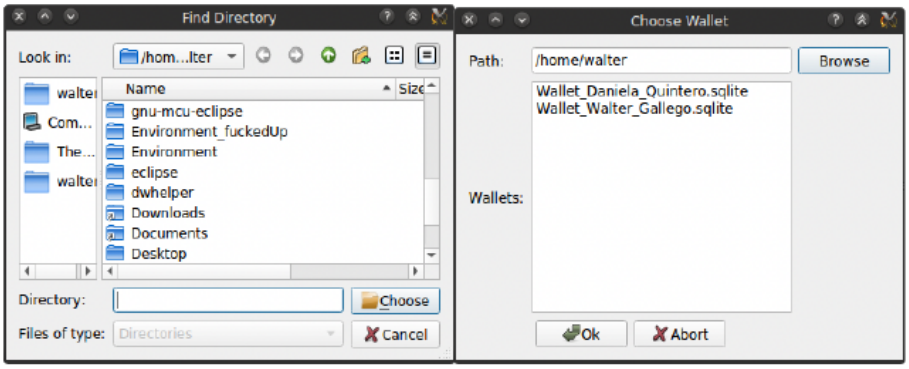

Figure 3: SEwalletGUI - open wallet dialog.

5.4. Adding a password to a wallet

In order to add a password to a wallet, the user must click on the 'New Entry' button. A new window will appear (Figure 4), asking to the user to fill a form containing mandatory information about the new password, including the password itself.
Notice that at the bottom of the window there is a bar reporting the security level of the password, this value is computed thanks to the zxcvbn password strength estimation tool. Similarly, there is also a button to generate automatically the password.

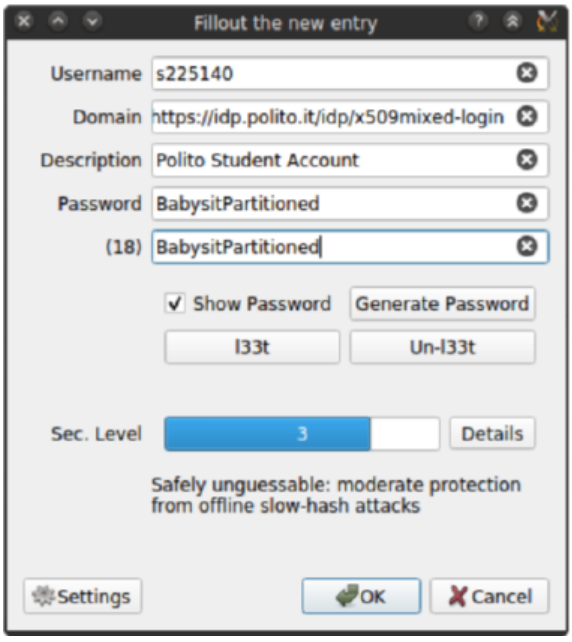

Figure 4: SEwalletGUI - dialog to add a new password.

6. Password and passphrase generation libraries

SE*wallet*™ has been developed including few libraries and tools that are useful to generate strong passwords and to estimate the strength of passwords. These tools can be configured by means of the **SE*wallet*™** GUI; when adding a new password to a wallet, the user can decide wether to use the tools to automatically generate a strong password or to enter the password manually. In this section, the tools included in **SE*wallet*™** are briefly described.

6.1. PwGen - pronounceable password generator

PwGen[7] is an open source program that generates secure, human friendly passwords. 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. These are the options available for **SE*wallet*™** users:

- *Length*. The desired length of the password. It is recommended to be at least 12 characters for non-random passwords and 8 characters 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 the usage 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 not to have to worry about random passwords accidentally containing offensive sub-strings.
- *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 one capital letter and one 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 easily managed via the **SE*wallet*™** application.

6.2. Custom passphrase generator

PwGen can generate pseudo-random pronounceable passwords, but these password are usually considered not strong enough when analyzed by password strength estimators. In order to solve this problem, a custom passphrase generator was developed inside **SE*wallet*™** ; it works by randomly picking words from dictionary files. The user can tune the passphrase generation as follows:

- *Dictionaries*. The user must select appropriate dictionaries, containing a sufficient number of lines (larger than 10000) to ensure that the picked words are really random. The English and Italian dictionaries used by the 'zxcvbn' password strength estimator are a good example. The user can work with as many dictionaries as desired, as long as each dictionary is formatted with one word per line.
- *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.

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

- *Minimum Length of Words*. With this option is possible to select only random words whose length is higher than a certain threshold. This is to make sure that the resulting passphrase is not too short and, therefore, not secure. The drawback 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 the 'zxcvbn' password strength estimator, ordering words by frequency (the least common words are placed at the end of the dictionary), this option enabled the generation of a passphrase containing only unusual words. The drawback is that there are fewer words to choose from; however, 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.

7. Password strength estimation tool: zxcvbn

For over thirty 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[8] is an alternative password strength estimator that is small, fast, and not harder than LUDS to adopt. Zxcvbn is regarded by the community as one of the most reliable and mathematically advanced open source password estimators.
Zxcvbn calculates the entropy of a password as the sum of its constituent patterns. Any gaps between matched patterns are treated as brute-force "patterns" that also contribute to the total entropy. Notice that calculating the entropy of a password as the sum of entropies of its constituent patterns is a conservative assumption, because it disregards the entropy given by all the possible combination of constituent patterns that can be created. Zxcvbn purposely underestimates, assuming that the attacker already knows the structure of the password, so the attacker does not need to perform a brute-force on the entire password but on subsets determined by the patterns. Currently, zxcvbn matches against:

- *Dictionaries*. Common words that the user is likely to choose as password; multiple dictionaries in *.txt* format can be used. Preloaded dictionaries include: English words, Italian words, names and surnames, Burnett's 10.000 common passwords, words from TV and movies. The match has an associated frequency rank, where words like 'the' and 'good' have a low rank, and words like 'photojournalist' and 'maelstrom' have a high rank. This lets zxcvbn scale the calculation to an appropriate dictionary size on the fly, because if a password contains only common words, an attacker can succeed with a smaller dictionary. For all dictionaries, match recognizes uppercase letters 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 2021 are considered, as well as dates in different formats (e.g., 3-13-1997, 13.3.1997, 1331997).

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

Appendices

A. Inner implementation

This section addresses few aspects of the inner implementation of **SE*wallet*™** , so that developers can better understand how to modify and extend the application.
SE*wallet*™ has been developed decoupling the interaction with the user from the interaction with the **SE*cube*™** . There are two applications in **SE*wallet*™** :

1. the GUI, that is implemented by means of the Qt Framework and that manages the interaction with the user;
2. the backend, that is a console application implemented in C++ that handles the interaction with the **SE*cube*™** device.

Moreover, there are other important elements such as the tools for password generation and password strength estimation. The overall architecture of the application is illustrated in Figure 5.

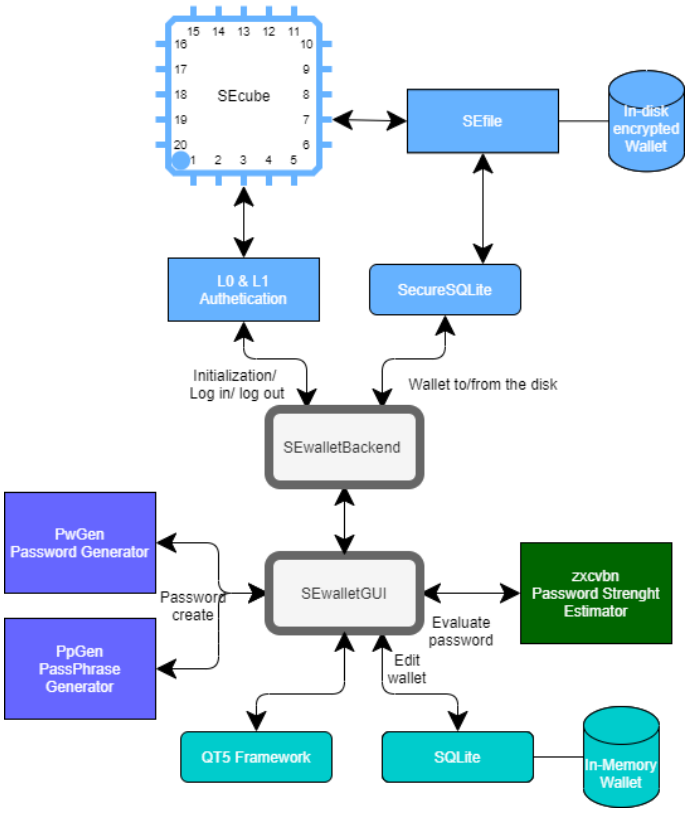

Figure 5: **SE*wallet*™** application architecture.

The backend and the GUI communicate by means of a TCP socket. This is the workflow of **SE*wallet*™** :

1. the user launches the GUI;
2. the GUI automatically launches the backend;
3. the backend listens for incoming TCP connections from the GUI;

4. the GUI connects to the backend;
5. the GUI sends various commands to the backend to execute different actions, such as searching for the **SE*cube*™** device;
6. the backend handles the requests coming from the GUI, forwarding them to the **SE*cube*™** ;
7. the GUI processes the responses coming from the backend, showing the required data to the user.

The choice of the TCP socket for the inter process communication was due to the fact that it was really fast and simple to implement. It is not secure, because a malware could sniff the communication on the socket; however, this approach was adopted because the backend will soon be replaced by a more complex software for the **SE*cube*™** that will include all the features of the current backend and that will support many other applications as well. The new software will not use standard TCP sockets but, in principle, the architecture of the solution will be the same, with two processes and an inter process communication protocol.
From an architectural point of view, the user never interacts directly with the backend, since the backend works as a dispatcher of requests from the GUI to the **SE*cube*™** (and vice versa for the responses). Moreover, **SE*wallet*™** includes other applications that are used to provide additional features, such as the generators of passwords and passphrases and the password strength estimator. Notice that these tools are logically connected to the Qt GUI, not to the backend.