Politecnico di Torino

Cybersecurity for Embedded Systems

SEcube Utilities: a GUI for the SEcube device

Project Report

Master degree in Computer Engineering

Referents: Prof. Paolo Ernesto Prinetto, Matteo Fornero, Nicolò Maunero, Gianluca Roascio

Gaetano Galasso (274472)
Andrea Pignata (280098)
Andrea Russo (277941)

September 29, 2021

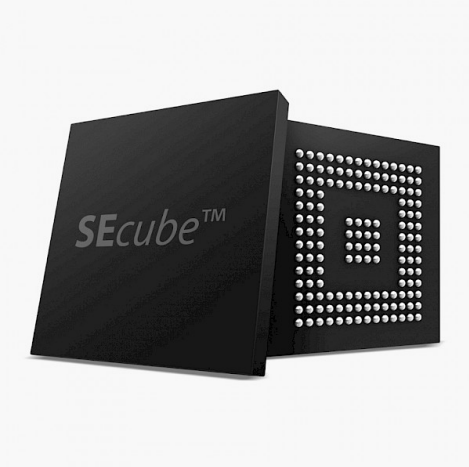
Contents

CHAPTER 1

Introduction

1.1 What is SEcube?

SEcube is the smallest reconfigurable silicon combining three main cores in a single-chip design. Low-power ARM Cortex-M4 processor, a flexible and fast Field-Programmable-Gate-Array (FPGA), and an EAL5+ certified Security Controller (SmartCard) are embedded in an extremely compact package. This makes it a unique security environment where each function can be optimised, executed, and verified on its proper hardware device. In a very small form factor (BGA 9x9mm) SEcube features all the useful standard interfaces, suitable for implementations in many and most devices. SEcube is based on a modular software architecture where all the functional blocks are isolated and well documented.



1.2 Overview

The SEcube utilities toolkit has been developed to overcome the greatest difficulty when working with the SEcube open source firmware: there was not an handy user interface to be used to perform basilar operations, like encryption, decryption or computing digest of files. Our goal is to develop an easy-to-use application that final users of the SEcube can install on their machines and use without the hassle of compiling small programs or using the command line.
At the current time, the application is compatible with Windows and Linux. It can:

- **Encrypt** files with the SEfile library, using one between the SEkey KMS or the manual key management system.
- **Decrypt** files in the same manner of encryption.
- Compute **digest** of any file.

The application can also be executed by right-clicking on a file and selecting the SEcube entry, so that the application will start with the path of the selected file already inserted in the "File" field (available on Windows only).
Moreover, the application can be executed either by command line or GUI.

1.3 Encryption utility

In cryptography, **encryption** is the process of encoding information. This process converts the original representation of the information, known as **plaintext**, into an alternative form known as **ciphertext**. Encryption does not itself prevent interference but denies the intelligible content to a would-be interceptor.
An encryption scheme usually uses a **pseudo-random encryption key** generated by an algorithm. It is possible to decrypt the message without possessing the key but, for a well-designed encryption scheme, considerable computational resources and skills are required. An authorized recipient (who receive the information) can easily decrypt the message with the key provided by the originator to recipients but not to unauthorized users.
There are two main types of keys in cryptographic systems, that are **symmetric-key** and public-key (also known as **asymmetric-key**): in the first one, the encryption and decryption keys are the same, and communicating parties must have the same key in order to achieve secure communication; in the second one, the encryption key is published for anyone to use and encrypt messages, but only the receiving party has access to the decryption key that enables messages to be read.

1.4 Decryption utility

In cryptography, **decryption** is the process of decoding information. This process converts the **ciphertext** into the original representation, known as **plaintext**. The header of the encrypted file contains information about encryption, so decryption process does not need to know input parameters like the key or the algorithm used.

1.5 Computing digest utility

A cryptographic **hash function** is a mathematical algorithm that maps data of an arbitrary size (often called the "message") to a bit array of a fixed size (the "hash value", "hash", or **"message digest"**). It is a **one-way function**, that is, a function for which is practically infeasible to invert or reverse the computation.

The ideal cryptographic hash function has the following main properties:

- It is deterministic, meaning that the same message always results in the same hash;
- It is quick to compute the hash value for any given message;
- It is infeasible to generate a message that yields a given hash value;
- It is infeasible to find two different messages with the same hash value;

- a small change to a message should change the hash value so extensively that a new hash value appears uncorrelated with the old hash value (**avalanche effect**).

An important application of secure hashes is the verification of message integrity. Comparing message digests (hash digests over the message) calculated before, and after, transmission can determine whether any changes have been made to the message or file.

CHAPTER 2

SEcube Utilities

2.1 Overview

The SEcube Utilites application is composed of two basic modules: a fully-featured command line program and a Grapich User Interface, both with these characteristics:

- Encryption: it is possible to encrypt a file using SEfile. The user can choose to use a key manually stored inside the SEcube or to rely on SEkey.
- Decryption: also here, the SEfile library is used and the user can choose between SEkey or manual keys.
- Digest computation, choosing between sha-256 (no key required) and hmac-sha-256 (key required). In the latest, one can choose to set manually the nonce used to compute the digest (e.g., if you want to compare the digest with a one already computed, knowing the nonce used), or letting the SEcube to generate it randomly.
- Some *general purpose* utilities: list of the keys, list of the connected devices, update of the SEkey path.

Thanks to the intuitive commands syntax and the presence of an help command, the command line program can be copied and used on any device, without the need of a desktop environment.

2.2 How to use

The compiled executable can be used inside a terminal window both in Linux and Windows. The command ***secube_utilities -help*** (see Chapter 5 for more information about the name of the command to use) will show a complete guide of all the functionalities offered by the executable (Fig. 2.1). The order of the argument is not important, but all the necessary ones should be written, else some default values will be used.

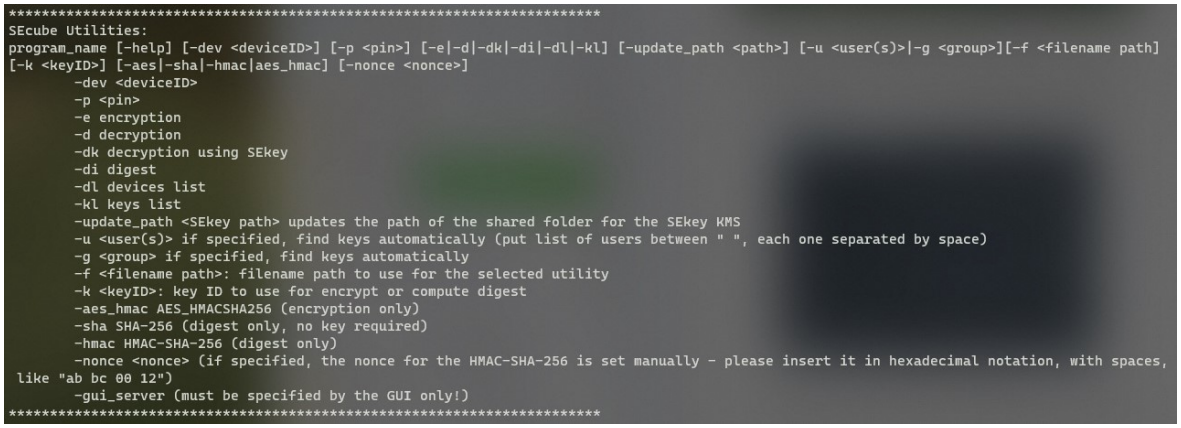


Figure 2.1: The help dialog of the backend.

For completeness, the following examples will show you how to use both command line and GUI.

- List of the connected devices: *"secube_utilities -dl"* on **command line**. Click on "List devices..." button on **GUI** (Fig.2.2). It can be done on all the tabs (encryption, decryption, etc.), and by clicking on one of the device in the list, the corresponding device ID will be inserted in the "Device ID" field.

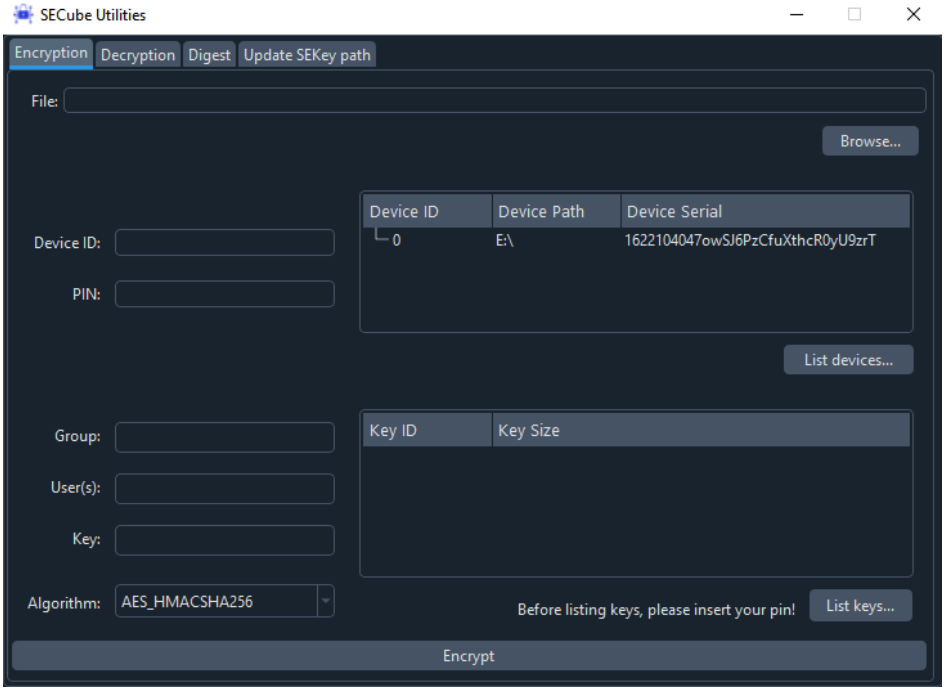


Figure 2.2: Device list command on GUI.

- List of the keys in device *0* with pin *test*: *"secube_utilities -kl -dev 0 -p test"* on **command line**. On **GUI** (Fig.2.3), once you inserted the device and pin, click on "List keys..." to see the keys contained in that device. As above, you can click on the desired key and it will be inserted in the "Key" field.

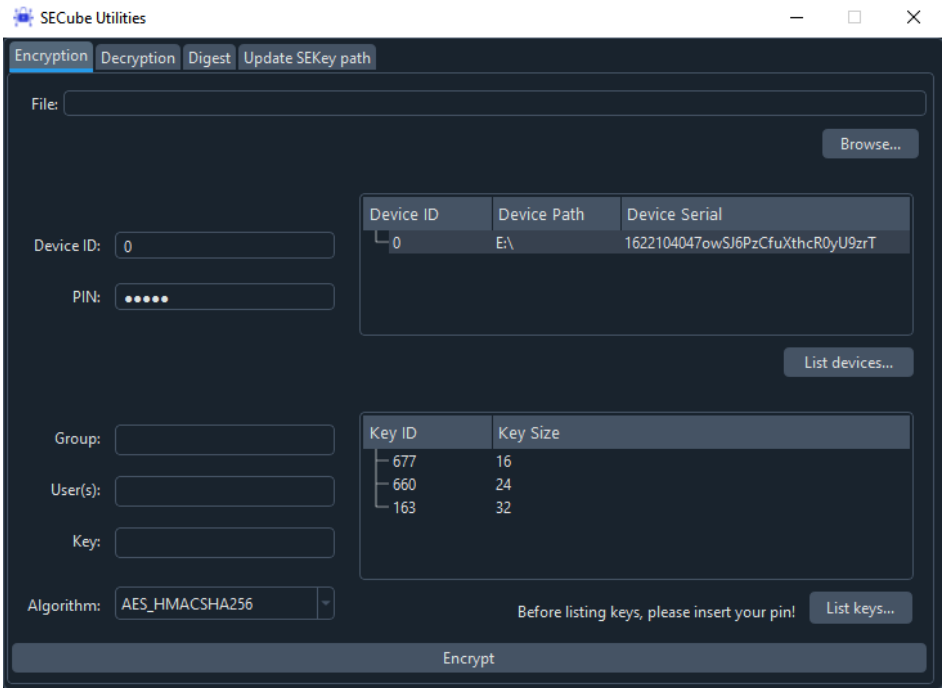


Figure 2.3: List keys command on GUI.

- Update SEkey path with device *0*, pin *"test"* and path *"new_path"*: *"secube_utilities -dev 0 -p test -update_path new_path"* on **command line**. On **GUI** (Fig.2.4), go on tab "Update SEKey path", choose the new path from the file explorer ("Browse..." button), then insert device and pin.

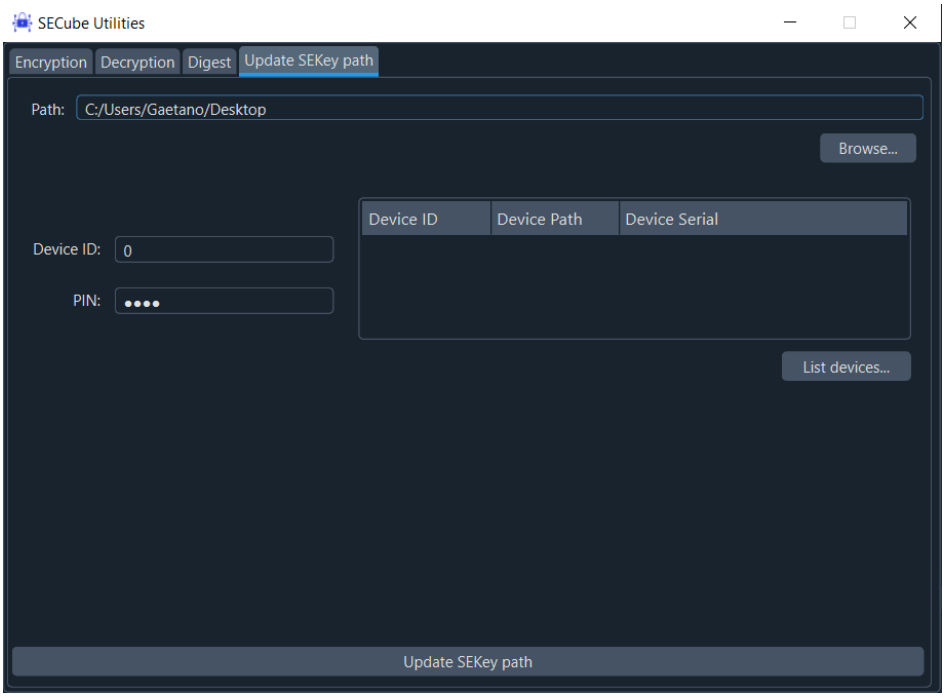


Figure 2.4: Update SEkey path on GUI.

- Encryption of a file *file.txt* using device *0*, key *500* and pin *"test"*: *"secube_utilities -e -dev 0 -p test -k 500 -f file.txt -aes_hmac"* on **command line**. On **GUI**, see Fig.2.5.

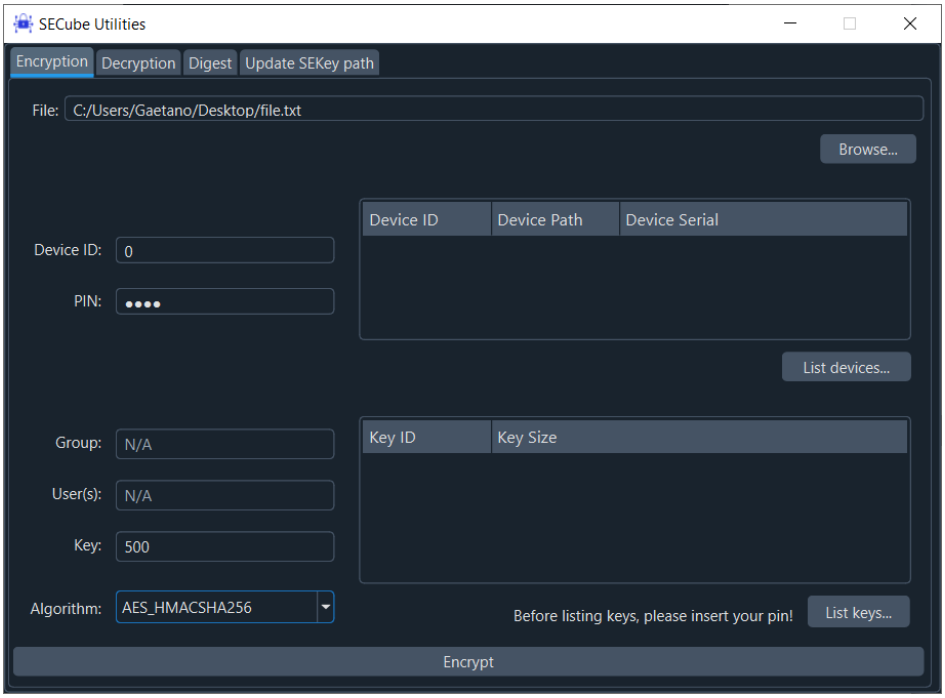


Figure 2.5: Manual encryption on GUI.

- Encryption of a file *file.txt* using SEkey (users U05, U06), with device *0* and pin *"test"*: *"secube_utilities -e -dev 0 -p test -u "U05 U06" -f file.txt -aes_hmac"* on **command line**. On **GUI**, see Fig.2.6.

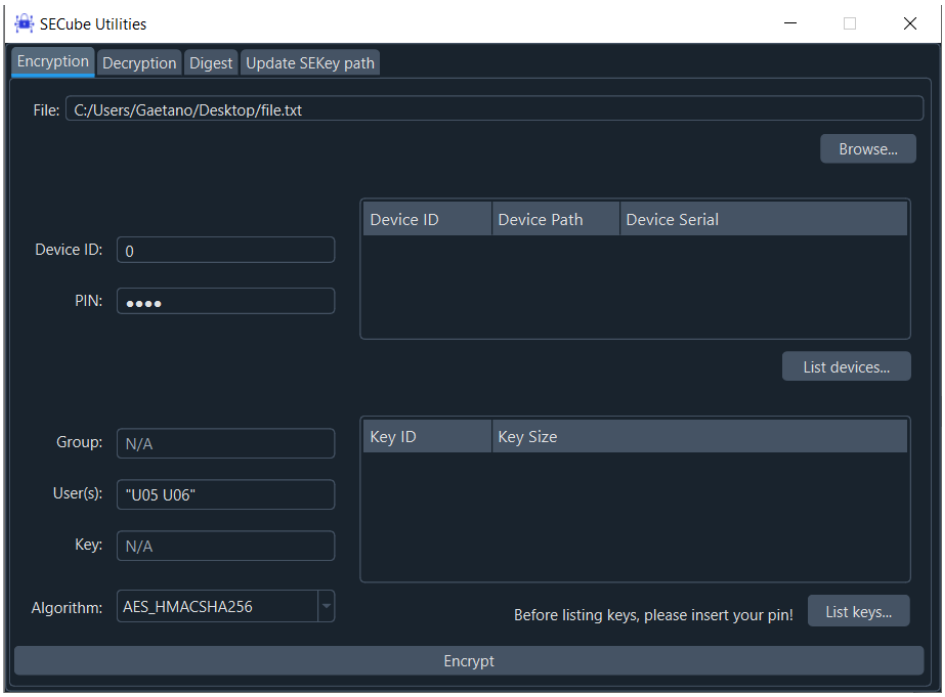


Figure 2.6: Encryption using SEKey on GUI.

- Decryption of a file *encryptedfilename.txt*, with device *0* and pin *test*: *"secube_utilities -d -dev 0 -p test -f encryptedfilename.txt"* on **command line**. On **GUI**, see Fig.2.7.

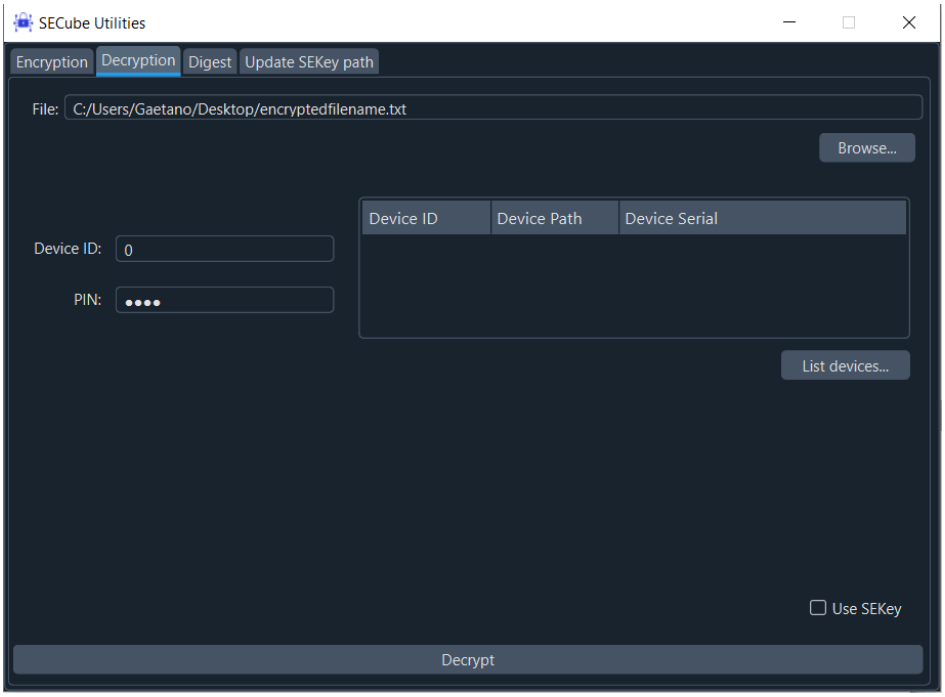


Figure 2.7: Decryption on GUI.

- Decryption of a file *encryptedfilename.txt* using SEkey, with device *0* and pin *test*: *"secube_utilities -dk -dev 0 -p test -f encryptedfilename.txt"* on **command line**. On **GUI**, see Fig.2.8.

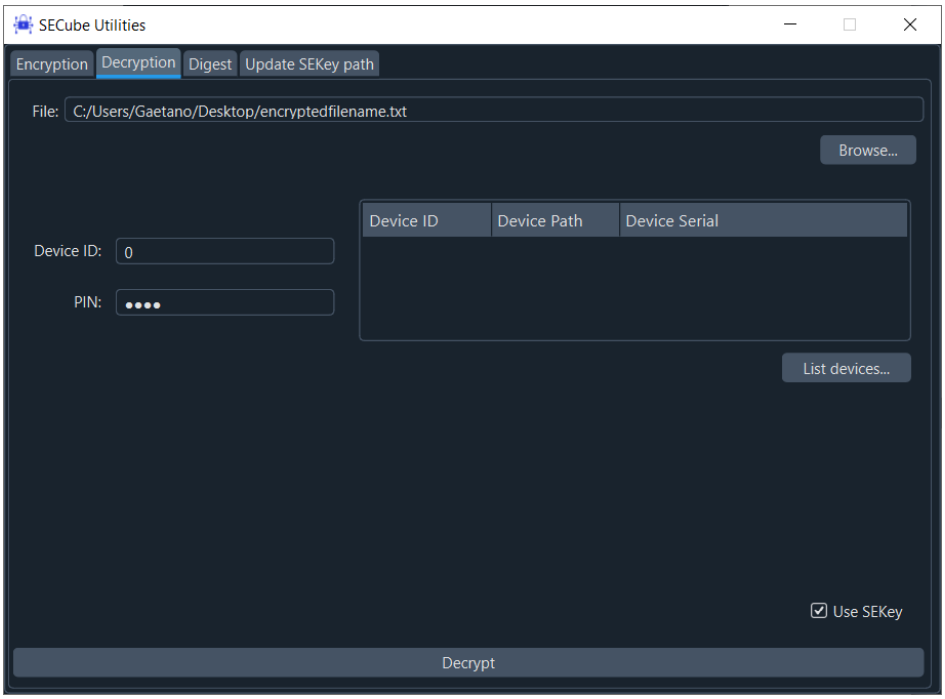


Figure 2.8: Decryption using SEKey on GUI.

- Digest computation of a file *file.txt*, with device *0* and pin *test*: "secube_utilities -di -dev 0 -p

test -f file.txt -sha" on **command line**. On **GUI**, see Fig.2.9.

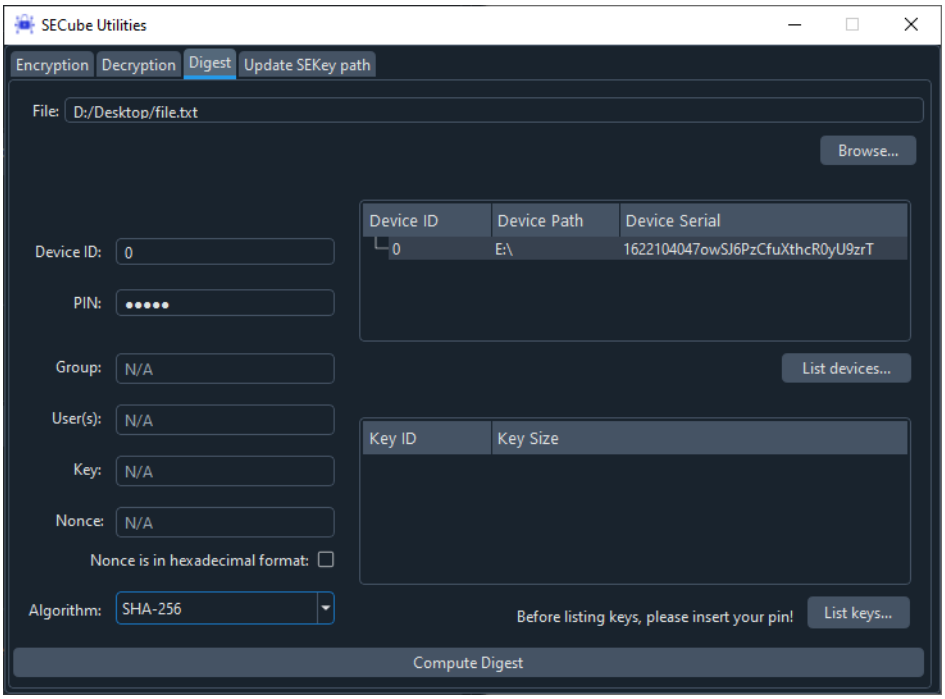


Figure 2.9: Computing digest on GUI.

- Digest computation of a file *file.txt*, with device *0*, key *500* and pin *test*: "secube_utilities -di -dev 0 -p test -k 500 -f file.txt -hmac" on **command line**. On **GUI**, see Fig.2.10.

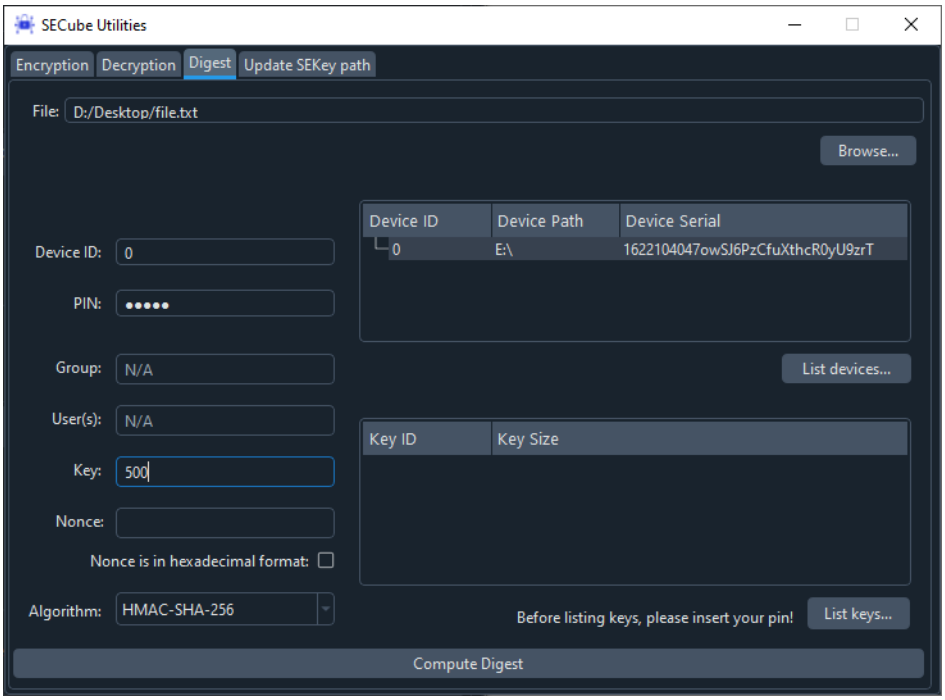


Figure 2.10: Computing digest using key on GUI.

- Digest computation of a file *file.txt*, with device *0*, key *500*, nonce *"use this nonce"* set manually and pin *"test"*: "secube_utilities -di -dev 0 -p test -k 500 -f file.txt - nonce "75 73 65 20 74 68 69 73 20 6e 6f 6e 63 65" -hmac" on **command line**. On **GUI**, see Fig.2.11. Please notice that the command line only supports hexadecimal input when inputting nonces, in order to avoid problems with some characters.

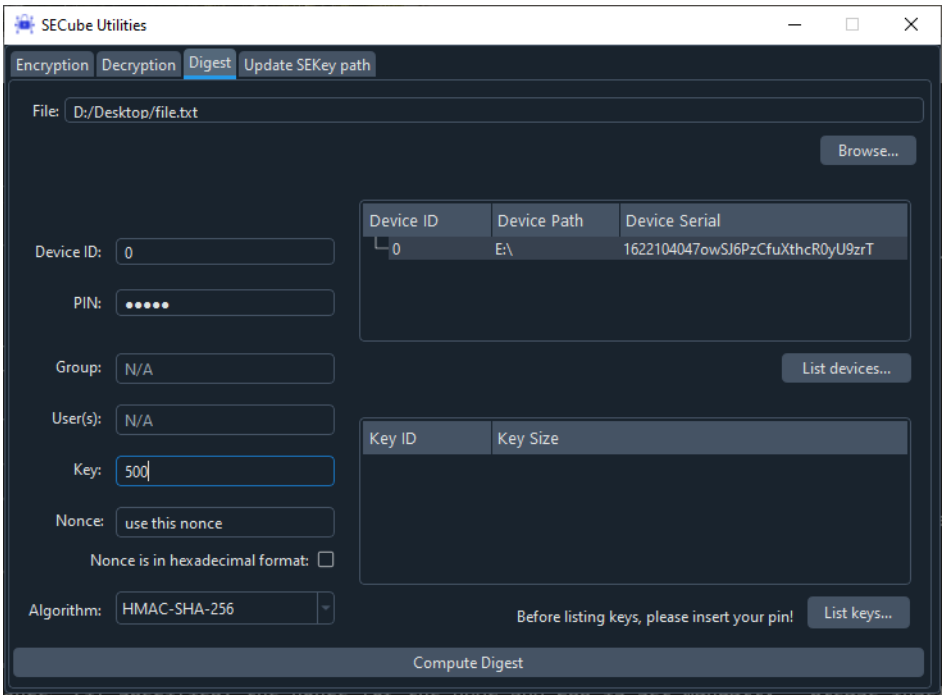


Figure 2.11: Computing digest using key and nonce set manually on GUI.

2.3 How to launch

In order to launch the application, the user can follow these steps:

- Windows, with installation: to install the application on the current system, a double click on the *install.bat* file in the *secube_installer* folder is enough. Of course, the user should click "yes" when elevate privileges are requested. To launch the application, a right click on any file will show up the *Secube* menu item.
- Windows, no installation: the user can go inside the folder *binaries* contained in *secube_installer*. Then, a double click on the *secube_utilities* executable will launch the application.
- Linux: a double click on the *secube_utilities.sh* executable, contained in the *secube_linux* folder, will launch the application. Remember to set all the executables with execution permissions, including *secube_cmd.exe* and *secube_utilities* in the bin folder.
- Compilation: the user can follow the instructions contained in the *Compilation Steps* chapter.

CHAPTER 3

Conclusions and Comments

3.1 Problems

During the designing of the application, we faced a problem concerning the decryption utility on Linux: in particular, the secure_ls() API (see Chapter 4 for more details about this function) does not work on Linux, so the decryption_w_encrypted_filename function cannot retrieve the original name of the file to decrypt; this issue does not depend on the implementation of the SEcube utilities program, so we do not focus on it too much time.

3.2 Conclusion and future works

We want to conclude this report talking about the possible improvements and commenting the overall project.
Possible improvements are:

- **Secure explorer**: since the encrypted files have the name encrypted too, we think could be useful a section in which one can see the encrypted files with the name already decrypted, so he can choose the file to decrypt without "getting crazy" with the encrypted names.
- **SEkey integration in GUI**: SEkey works with users and groups, so we think could be useful a section in the GUI in which one can choose the user(s) and the group(s) that are eligible for encryption and digest computation, so that one does not need to remember who is able to decrypt or to receive information with him.

In conclusion, SEcube utilities is an easy-to-use program to encrypt, decrypt and compute digest, with some other features that help the user to easily use the three main utilities. For sure, it can be improved in several ways (some of them are listed above), but we are happy about the final product. Improvements in the SEcube SDK and APIs could make even better the application, having more algorithms to use and some other functionalities that can be added.

CHAPTER 4

Implementation details and APIs

The project is composed of these folders:

- Documentation, the folder containing this LaTeX report, the PowerPoint file for the project presentation and a *.mp4* video demonstrating how the application works.
- secube_utilities_backend, containing the source files for the backend compilation
- secube_utilites, containing the source files for the GUI compilation.
- secube_installer, containing a Windows installer for the application,
- secube_linux, containing the application already compiled for Linux.

4.1 Backend Details

The project is composed of some folders:

- sources, sefile, sekey, sqlite contain the source code of the SEcube open SDK.
- Src, Inc contain respectively the source code and the include file we wrote to implement all the functions of the backend.
- cereal contains all the source code for the Cereal library, that will allow the communication between the GUI and the backend through sockets.

Inside the Src directory, the source code is divided into these files:

- SCU.cpp contains the main() function, that parses all the arguments passed via command line and calls the correct function to perform the required operation.
- utilities.cpp contains all the functions used to perform the cryptographic operations:
  - The *digest* function computes the digest of a particular file. It requires some parameters: the filename and the algorithm are compulsory, while the key is required when calculating an authenticated digest (via the HMAC-SHA-256 algorithm).
  - The *encryption* function encrypt a file using SEfile. The required parameters are the filename, the ID of the key and the algorithm.

  - The *decryption* function decrypts the file passed as argument using SEfile. The filename should be the decrypted one, as the secure_open() function of the SEcube SDK requires that filename. This particular function is not used in the final version of the program, but we left it there to eventually expand the functionalities in the future.
  - The *decryption_w_encrypted_filename* does the same as the other one, but accepts the encrypted filename. It calls the secure_ls() function of the SEcube SDK to obtain the decrypted filename, and then proceed as the other function. We decided to use this function as the decryption one because, when working with a GUI with a file browser, we need to work with filenames that actually exist.
  - *list_devices* returns a list of all the connected SEcube devices.
  - *list_keys* returns a list of all the keys added to the SEcube device in manual key management mode.
  - *login* performs the login to the SEcube device, updating the L0 and L1 pointers.
  - *logout* performs the logout from the SEcube device.
  - *find_key* can query the SEkey KMS for a key that is shared with a particular user, with more users or with a group. It accepts as arguments a pointer to the key, a string of user(s) and a string with the group. On the base of the length of the passed strings, it can understand which particular SEkey function should be called.

- GUI_interface.cpp and linux_GUI_interface.cpp are the source code files that implement the communication between the backend and the GUI through inter-process sockets. The first one is compiled on Windows devices, the second one on Linux machines.

Some functions may need to contact the GUI through sockets (and cereal), in that case the socket should be passed by argument if the backend is started in gui mode (through the flag -gui_server). Each function can verify the value of the global variable *gui_server_mode* to understand if the sockets should be used or not.

4.2 GUI Details

The GUI has been designed by means of the QT Designer application. Hence, some files has been written automatically by the tool. Some details will follow:

- *secube_utilities.pro* contains all the details of the project, and it should be tuned with some details of the compiler as described in the Compilation Steps.
- *utilities.cpp* and its header file contains all the functions that are called when the user interacts with the GUI (for example when a button is clicked).
- *utilities.ui* contains all the information about the design of the UI, and has been generated automatically by the QT Designer tool.
- *main.cpp* contains the code to create the application GUI as soon as it's started. This file has been generated automatically by the tool.
- *backend_interface.cpp* and *linux_backend_interface.cpp* are the two source files that implement the communication between the backend and the GUI through inter-process sockets.
- The *cereal* folder contains the source files for the Cereal library we implemented to make socket communication easier, the *logo* folder contain graphical resources for the app and the *qdarkstyle* contains resources to implement the dark style theme we applied to the application.

4.3 Windows Installer

This application can be used standalone, but we decided to also offer a Windows installer: this is the only way to offer the contextual menu functionality, as the executable file of the GUI should be available in a particular folder that the system knows a priori. The installation also offer the possibility to call the backend from any location on the system with the command line (it can be useful if anything happens on the machine and mainteinance is necessary).

After double-clicking on the installer (a .bat file), the following operations are done:

- **The program installation directory is added to the global PATH variable**. In this way, the system understand that executables can be found in that directory and that they can be called from everywhere. This is needed to allow the backend to be executed anywhere it is requested, from the GUI or the command line.
- **The installer requires administrator privileges**. This is necessary in order to copy the files in the Program Files directory.
- **Files are copied in the Program Files directory**. This is done by using the %Program-Files% alias of the Windows command line and the xcopy command, that allows the copy of entire directories. Please notice that the files that will be copied inside the Program Files are the one found in the *binaries* folder of the SEcube installation folder.
- **The shortcut is added to the contextual menu**. This is done by adding a key to the Windows system register.

CHAPTER 5

Compilation Steps

The SEcube utilities package comes with a Windows installer and a portable application for Linux that are already compiled. If the user wants to compile the entire application on its own, he can proceed as follows.

5.1 Backend

The SEcube application is composed of two components: the GUI and the backend. As the GUI needs the backend in order to communicate with the SEcube, it is necessary to compile it first.

The necessary tools to compile it are the *Eclipse for C/C++ Developers* IDE and a valid compiler for C++, like *g++*. Eclipse can be downloaded from its website (www.eclipse.org) for free. Usually g++ is already installed on linux distributions, while Windows users may need to install the *minGW64* package. Ubuntu users can install it by running *sudo apt-get install build-essential* on a terminal window.

1. First of all, create a new C/C++ project on Eclipse.
2. Go to File - Import - General - File System and click Next. Click Browse, and choose the *secube_utilities_backend* folder, then import everything from it.
3. Only on Windows: Go to Project - Properties - C/C++ Build - Settings - MinGW C++ Linker. On the *Command* line, edit the text to be *g++ -static -static-libgcc -static-libstdc++*. This is necessary to create a standalone .exe file that does not require any external library.

   Then, choose *Miscellaneous* under the C++ Linker, and under *Other Objects* add the *libws2_32.a* file you can find in the mingw64 installation path, in the *lib* folder.
4. Set the compiler to use C++ 17 by going to Project - Settings - C/C++ Build - Settings - GCC compiler - Dialect.
5. Now everything is ready to be compiled. Go to the Project menu, and click Build All.
6. Go inside your Eclipse workspace folder, and identify the executable file you just compiled. It is available in the Debug (or Release) folder. This file is particularly important when building the GUI.

5.2 GUI Interface

After the backend executable file has been generated, the user can proceed and compile the graphical interface.

To do so, the QT package should be installed. The open source package can be downloaded for free from their website (www.qt.io). During the procedure make sure that the compiler, the QT base package and the QT creator application are chosen for installation. After everything is ready, proceed as follows.

1. Open QT creator. In the Welcome page, choose to open an existing project, and locate the *secube_utilities.pro* file in the *secube_utilities* folder. If there is any problem due to the configuration that can't be loaded, don't worry and press "Configure Project", then take a look at the following steps.

2. Once the project is loaded, open the secube_utilities.pro file and verify the latest lines. If you're on Linux, comment any line starting with LIBS. If you're on Windows, leave the *-lwsock32* line as it is and replace the other line with the correct path of the *libws2_32.a* file. The file can be found in the MinGW64 installation path or in the QT installation path.

3. On Linux, add the line *QMAKE_LFLAGS += -no-pie* in the secube_utilities.pro file. In this way, a standalone executable will be created, and it can be launched with a double click.

4. Then, click on Project on the bar on the left. In the *Build* section, choose as the Build Directory the secube_utilities folder you found the .pro file in. In the *Run* section, verify the path of the Working Directory and annotate it.

5. Now, go into the Working directory found at the previous step and copy inside of it the executable file of the backend built before. **Important:** rename it as *secube_cmd.exe*, for both Windows and Linux. If you want to modify how the backend is called, its path or its name, you can modify the *system()* call in the *linux_backend_interface.h* file for Linux or the *CreateProcessA()* call in the *backend_interface.h* file for Windows.

6. Now the program can be launched by pressing the green "play" button on the left.

CHAPTER 6

Appendix

[1] SEcube website
https://www.secube.blu5group.com/
[2] Encryption utility
https://en.wikipedia.org/wiki/Encryption#Encryption_in_cryptography
[3] Digest utility
https://it.wikipedia.org/wiki/Funzione_crittografica_di_hash
[4] GUI dark style
https://github.com/ColinDuquesnoy/QDarkStyleSheet