SEfile™ Command Line Interface (SEfile-cli)

Project documentation

a.a. 2018 – 2019

Daniele Castro S253244

Release: June 16th, 2019

Proprietary Notice

The following document offers information, which is subject to the terms and conditions described hereafter.

While care has been taken in preparing this document, some typographical errors, error or omissions may have occurred. I reserve the right to make changes to the content and information described herein or update such information at any time without notice. The opinions expressed are in good faith and while every care has been taken in preparing this document.

Author

Daniele CASTRO danielecastro@hotmail.it

Trademarks

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by Blu5 View Pte Ltd. Other brands and names mentioned herein may be the trademarks of their respective owners. No use of these may be made for any purpose whatsoever without the prior written authorization of the owner company.

Disclaimer

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN “AS IS” BASIS AND ITS AUTHORS DISCLAIM ALL WARRANTIES, EXPRESS, OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY TAHT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PURPOSE.

THE SOFTWARE IS PROVIDED TO YOU “AS IS” AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEREUNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY LIKELIHOOD OF SUCH DAMAGES.

Table of contents

1. Features

This chapter presents the features of SEfile-cli Command Line Interface tool. The idea is to create a powerful tool that allows SEfile™ APIs to be used directly from linux bash or windows command prompt or within a script file without writing a single line of C code. It is also a very well-done usage example for those who wants to learn how to use SEfile™ APIs since it is very well commented:

- ☐ SEfile-cli supports many commands
- ☐ Each command does tasks that are most probably asked to be done by these APIs
- ☐ Each command supports a certain number of options
- ☐ Depending on the command some options are mandatory
- ☐ Not mandatory data options will be asked at runtime.
- ☐ Code is particularly well commented for those who wants to learn how to use these APIs.
- ☐ In each function prototype there is a doxygen comment explaining each single parameter.

2. What is the project thought to be for?

We initially thought to modify the current graphical telegram client to use SEcube as crypto engine upon the already existing encryption mechanism of telegram.
A similar project was already done in the past by other students but, since it was not using the standard telegram libraries, Tdlib, it does not compile anymore. Unfortunately, we immediately noticed that Tdlib does not provide any graphical API. That means that an entire new graphical client must be developed in order to use these libs.
So, we came up with another solution:
we decided to make telegram-cli able to use SEcube as crypto engine instead of telegram. To do so we decided not to modify the source code of telegram-cli. Instead, we decided to make an external C-written program that will be opportunely called by telegram-cli to encrypt or decrypt messages.
This is possible thanks to the fact that telegram-cli integrates the possibility to write LUA scripts that can consequently call external programs.
This C-written program is called “SEfile-cli”. It is a command line interface that interacts with the SEcube MCU to crypt or encrypt binary files.
It works both in Linux and Windows. Making a command line interface means that it can be used in scripts or by other programs. That makes this program a powerful command line tool. It can also be called, for instance, as a cgi-bin script from any web-browser to encrypt HTML passwords and so on.
Unfortunately, after making everything, I discovered that the development stage of telegram-cli is not giving a final stable software release. So, basically, the LUA script is not yet able to properly execute my executable. In the future developing stages of telegram-cli probably a few bugs will be solved and telegram-cli will be able to properly run my executable.

3. What is SEfile™?

SEfile™ is a software library made to interact with the SEfile™ firmware to be flash on the SEcube™ microcontroller. This firmware and library allow to use the crypto-engine peripheral on the SEcube™ microcontroller to crypt files on the PC filesystem.
It is a library which exploits the hardware key management exposed from APIs Level L1 and other functionalities from the SEcube™ device. It has been developed having in mind the needs to ensure both simplicity of usage and security for data at rest: it allows secure storage, retrieve and usage of information that could not be trusted if stored elsewhere, e.g., any personal computer, or cloud service provider.

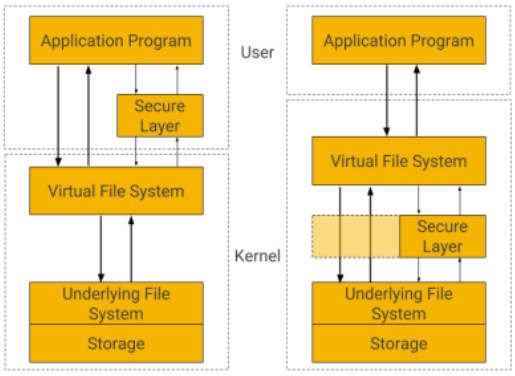

Figure 1 – Secure Layer and Virtual File System: two different approaches

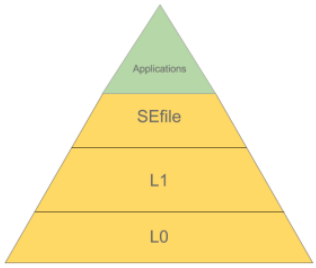

Figure 2 – **SE*file*** hierarchic organization

Conceptually, SEFile™ targets any user that, by moving inside a secure environment, wants to perform basic operation on regular files. It must be pointed out that all encryption functionalities are demanded to the secure device in their entirety. In addition, SEFile™ does not expose to the host device details about what, or where it is reading/writing data: thus, the host OS, which might be untrusted, is totally unaware of what it is writing.

4. How to compile

The project is a simple CMake one. You run CMake in the project directory and after you run Make. At the end of the compilation you will find the SEfile-cli executable in the root directory of the project. Here is a small script that can be run to compile this project in Linux:

```
git clone https://gitlab.com/DanCaster/setelegram.git
cd setelegram
```

```
cmake .
make
./SEfile-cli
```

Some CMake-ready IDE like JetBrains CLion can automatically recognise the CMakeLists.txt and compile the program without edit any setting. Alternatively, CMake can be run from a console and, once a Make file is generated, any IDE can compile everything.

5. Code architecture

5.1. Global architecture

The code is structured in three levels:

- ☐ Command Line Interface (CLI) C functions
- ☐ Wrapper C functions
- ☐ SEfile™ APIs

It is a CMake development project.

5.2. Command Line Interface (CLI) C functions

The Command Line Interface functions simply implements the functions that can be requested by the CLI interface with commands and parameters.

They are:

void list (char * ***peripheral*****, char *** ***password*****, char *** ***directory*****)**

This function Shows a list of decrypted file names of encrypted files in a given directory.

Parameters:

in	**peripheral*	is the windows drive letter (ex: D) or partition path for Linux.
in	**password*	is the SEfile firmware password
in	**directory*	is the directory path

int main (int ***argc*****, char *** ***argv*****[])**

This the main function.

Parameters:

in	*argc*	is the number of arguments passed to the program
in	**argv[]*	is a string array containing the arguments string

void wrcff (char * ***peripheral*****, char *** ***password*****, char *** ***file_path*****, char *** ***cipher_file_path*****)**

This function writes a cipher file starting from a binary file: it crypts the content of the binary file into a cipher one.

Parameters:

in	**peripheral*	is the windows drive letter (ex: D) or partition path for Linux.
in	**password*	is the SEfile firmware password
in	**file_path*	is the string containing the binary file path
in	**cipher_file_path*	is the string containing the cipher file path

void wrcfs (char * *peripheral*, char * *password*, char * *string*, char * *cipher_file_path*)

This function writes a cipher file from a string: it crypts the content of the string into a cipher file.

Parameters:

in	**peripheral*	is the windows drive letter (ex: D) or partition path for Linux.
in	**password*	is the SEfile firmware password
in	**string*	is the input string
in	**cipher_file_path*	is the string containing the cipher file path

void wrffc (char * *peripheral*, char * *password*, char * *file_path*, char * *cipher_file_path*)

This function writes a file from a cipher one: decrypts the content of the cypher file into a binary file.

Parameters:

in	**peripheral*	is the windows drive letter (ex: D) or partition path for Linux.
in	**password*	is the SEfile firmware password
in	**file_path*	is the string containing the binary file path
in	**cipher_file_path*	is the string containing the cipher file path

void wrsfc (char * *peripheral*, char * *password*, char * *cipher_file_path*)

This function writes a string from a cipher file: decrypts the content of the cypher file into a string.

Parameters:

in	**peripheral*	is the windows drive letter (ex: D) or partition path for Linux.
in	**password*	is the SEfile firmware password
in	**cipher_file_path*	is the string containing the cipher file path

They can all be found in “SEfile-cli.c” and “SEfile-cli.h”.

They are the only ones that are not commented in the prototype since they exactly do what described in the help message printed by `help()`.

5.3. Wrapper C functions

The Wrapper functions are a middleware between the APIs and the CLI.I made them because many common tasks that are usually asked to these APIs like “write a cipher file from a given binary file” or “write an encrypted buffer starting from a cipher file” can, now, be done calling a single function instead of coping and pasting every time few lines of code.

They are:

se3_disco_it choose_devices (char * *drive*)

This function chooses a device given an ASCII capital letter ex: D, E in Windows or the mount point in linux. In Linux only already mounted devices will be shown.

Parameters:

in	*drive*	letter in Windows, ex: D, E or the mount point in linux.

Returns:

The function returns a se3_disco_it Discovery iterator data structure.

void close_device (se3_session * *s*)

This function closes the SEcube USB connection. directory with decrypted file names.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**

se3_session init_device (char * *password*, se3_disco_it *it*)

This function handshakes the SEcube USB connection with the PC.

Parameters:

in	**password*	is the pointer to an already allocated ASCII string containing the device password.
in	*it*	is the Discovery iterator data structure initialized by **show_and_choose_devices()** or **choose_devices()**

Returns:

The function returns a SEcube Communication session structure.

void list_cipher_files_in_directory (char * *path*)

This function lists all the encrypted files in the directory with decrypted file names.

Parameters:

in	**path*	is the ASCII path where to list the encrypted files.

SEFILE_FHANDLE open_cipher_file (se3_session * *s*, char * *file_path*, int32_t *mode*, int32_t *creation*)

This function opens the encrypted file.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**
in	**file_path*	is the pointer to an already allocated ASCII string containing the cipher file path.
in	*mode*	The mode in which the file should be created. See Mode_Defines.
in	*creation*	Define if the file should be created or it should already exist. See Access_Defines.

Returns:

The SEFILE_FHANDLE data structure used to access encrypted filesin which the file handle to the new opened file is placed after a success, NULL in case of failure.

se3_disco_it show_and_choose_devices (void)

This function prints a list of available devices ex: D, E in Windows or the mount point in linux. In Linux only already mounted devices will be shown.

Returns:

The function returns a se3_disco_it Discovery iterator data structure.

void write_binary_file_from_cipher_file (se3_session * *s*, FILE * *fd*, SEFILE_FHANDLE * *sefile_file*)

This function reads an encrypted file and writes a decrypted version.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**
in	**fd*	is the binary file pointer.
in	**sefile_file*	is the SEFILE_FHANDLE data structure used to access encrypted filesin which the file handle to the new opened file is placed after a success, NULL in case of failure.

int write_buffer_from_cipher_file (se3_session * *s*, uint8_t * *buffer*, int *buffer_len*, SEFILE_FHANDLE * *sefile_file*)

This function reads an encrypted file and writes a decrypted buffer.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**

out	**buffer*	is the binary buffer pointer.
in	*buffer_len*	is the binary buffer lenght.
in	**sefile_file*	is the SEFILE_FHANDLE data structure used to access encrypted filesin which the file handle to the new opened file is placed after a success, NULL in case of failure.

Returns:

The the number of bytes ridden.

void write_cipher_file_from_buffer (se3_session * *s*, uint8_t * *buffer*, int *buffer_len*, SEFILE_FHANDLE * *sefile_file*)

This function reads a binary uint8_t buffer and writes an encrypted file.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**
in	**buffer*	is the binary buffer pointer.
in	*buffer_len*	is the binary buffer lenght.
in	**sefile_file*	is the SEFILE_FHANDLE data structure used to access encrypted filesin which the file handle to the new opened file is placed after a success, NULL in case of failure.

void write_cipher_file_from_file (se3_session * *s*, FILE * *fd*, SEFILE_FHANDLE * *sefile_file*)

This function reads a binary file and writes an encrypted version.

Parameters:

in	**s*	is the SEcube Communication session structure initialized by **init_device()**
in	**fd*	is the binary file pointer.
in	**sefile_file*	is the SEFILE_FHANDLE data structure used to access encrypted filesin which the file handle to the new opened file is placed after a success, NULL in case of failure.

They can be found in “wrapper.c” and “wrapper.h” and are made so that most of the things you can expect to do with SEfile APIs can, now, be done with fewer lines of code without directly calling the APIs.

6. Comment organization

6.1. Overview

This project as not only the purpose to be a command line interface for the Sefile APIs. It is also a well-written code example for those who wants to integrate these APIs in their own projects.

Comments are:

- ☐ In each C function prototype in doxygen style
- ☐ Inside C functions

6.2. Function prototype doxygen style comments

Each function prototype in header files is well commented in doxygen style according to the SEfile™ APIs style.

```
/**
 * @brief This function reads a binary file and writes an encrypted
 *        version.
 * @param [in] *s is the SEcube Communication session
 *         structure initialized by \ref init_device()
 * @param [in] *fd is the binary file pointer.
 * @param [in] *sefile_file is the SEFILE_FHANDLE data structure
 *        used to access encrypted files @hideinitializer in which
 *        the file handle to the new opened file is placed after a
 *        success, NULL in case of failure.
 */
void write_cipher_file_from_file(se3_session *s, FILE *fd, SEFILE_FHANDLE *sefile_file);
```

Figure 3 – Function prototype comment

`@brief` Describes what the function does.

`@param` can be `[in]` or `[out]` depending weather if the parameter will be only ridden or written by the function. In that case will respectively be an input or an output.

`@return` when present, explains what the function returns.

6.3. Comments inside functions

Between code lined many comments have been put to increase readability and code understanding:

```
void wrcff(char *peripheral, char *password, char *file_path, char *cipher_file_path)    {
    //device opening
    se3_disco_it it;
    if(peripheral == NULL)    it = show_and_choose_devices();
    else    it = choose_devices(peripheral);
    if(password == NULL) {
        printf("Enter the device password: ");
        char psswrd[32];
        scanf("%s", psswrd);
        password = psswrd;
        fflush(stdin);
    }
    se3_session s = init_device(password, it);
    //cipher file creation & opening, file opening
    FILE *fd = fopen(file_path, "rb");
    SEFILE_FHANDLE sefile_file = open_cipher_file(&s, cipher_file_path, SEFILE_WRITE, SEFILE_NEWFILE);
    //cipher file writing
    //printf("writing cipher file...\n");
    write_cipher_file_from_file(&s, fd, &sefile_file);
    //closing files
    fclose(fd);
    secure_close(&sefile_file);
    //closing device
    close_device(&s);
}
```

Figure 4 – Function comments inside functions

Also, some debug messages are commented. They can be opportunely uncommented do debug the specific code slice.

7. Command Line Interface

Command Line Interface has been designed as simple as possible and in pure Linux style. No options abbreviations are available till now.

Typing 'SEfile-cli --help' will display a message well explaining how to properly use the program with many usage examples:

```
C:\Users\danie\CLionProjects\setelegram\cmake-build-debug\SEfile-cli.exe --help
Usage: SEfile-cli command [options]

SEfile-cli is a commandline tool to manage encrypted files
using SEcube microcontroller as crypto-engine.

commands:
  list   - list decrypted file names of encrypted files
  wrcff  - writes a cipher file from a file
  wrcfs  - writes a cipher file from a string
  wrffc  - writes a file from a cipher one
  wrsfc  - writes a string from a cipher file
  --help - shows this message

options:
  -pe - device drive letter for windows (ex: "D") or path for linux
  -i  - input file or string
  -o  - output file
  -c  - input or output cipher file
  -pa - device password
  -d  - directory to list

usage examples:
  SEfile-cli wrcfs -pa test -i "Hello world!" -c cipher_file_out.txt
  on Windows: SEfile-cli wrcff -pe D -pa test -i text_file_in.txt -c cipher_file_out.txt
  on Linux: SEfile-cli wrffc -pe /mnt/sdb1 -pa test -o text_file_out.txt -c cipher_file_out.txt

For all the commands if -pe is not specified all the available
devices will be displayed and one of them must be chosen.

For all the commands if -pa is not specified a password will be
asked.

When no string with -i is passed to wrcfs, it will expect one from
the standard input.

Written by Daniele Castro on 04/30/2019.
```

Figure 5 – Help message

7.1. Usage

There are six commands and six options. Depending on the chosen command some options are mandatory. Not mandatory data options will be asked at runtime.

Example of a non-mandatory option:
When writing `SEfile-cli wrcfs` if no `-i` is specified, an input string will be expected from the standard-input.
Example of a mandatory option:
When writing `SEfile-cli wrcff` if no `-i` is specified, no input files will be found, and the application will fail. Not putting mandatory options ends in an error message and the immediate program exit.

Usage examples:
```
SEfile-cli wrcfs -pa test -i "Hello world!" -c cipher_file_out.txt
```

on Windows: `SEfile-cli wrcff -pe D -pa test -i text_file_in.txt -c cipher_file_out.txt`

on Linux:
```
SEfile-cli list -pa test -d /home/User

SEfile-cli wrffc -pe /mnt/sdb1 -pa test -o text_file_out.txt -c cipher_file_out.txt
```

7.2. Program commands

There are six commands:

- ☐ `list:` list cipher files with decrypted names in a path
- ☐ `wrcff:` writes a cipher file from a file
- ☐ `wrcfs:` writes a cipher file from a string
- ☐ `wrffc:` writes a file from cipher one
- ☐ `wrsfc:` writes a string from cipher file
- ☐ `--help:` prints usage informations.

7.3. Command options

There are six options:

- ☐ `-pe:` stands for 'peripheral': drive letter on Windows or partition path in Linux
- ☐ `-i:` stands for 'input file path/string'
- ☐ `-o:` stands for 'output file path'
- ☐ `-c:` stands for 'cipher file path'
- ☐ `-pa:` stands for 'password'
- ☐ `-d:` stands for 'directory path to list'