Politecnico di Torino

Cybersecurity for Embedded Systems
01UDNOV

Master's Degree in Computer Engineering

Project Title

Project Report

Candidates:
Name Surname (student ID)
Name Surname (student ID)
Name Surname (student ID)

Referents:
Prof. Paolo Prinetto
Dr. Matteo Fornero
Dr. Vahid Eftekhari

Contents

List of Figures

List of Tables

Abstract

This is the space reserved for the abstract of your report. The abstract is a summary of the report, so it is a good idea to write after all other chapters. The abstract for a thesis at PoliTO must be shorter than 3500 chars, try to be compliant with this rule (no problem for an abstract that is a lot shorter than 3500 chars, since this is not a thesis). Use short sentences, do not use over-complicated words. Try to be as clear as possible, do not make logical leaps in the text. Read your abstract several times and check if there is a logical connection from the beginning to the end. The abstract is supposed to draw the attention of the reader, your goal is to write an abstract that makes the reader wanting to read the entire report. Do not go too far into details; if you want to provide data, do it, but express it in a simple way (e.g., a single percentage in a sentence): do not bore the reader with data that he or she cannot understand yet. Organize the abstract into paragraphs: the paragraphs are always 3 to 5 lines long. In LaTeXsource file, go new line twice to start a new paragraph in the PDF. Do not use to go new line, just press Enter. In the PDF, there will be no gap line, but the text will go new line and a Tab will be inserted. This is the correct way to indent a paragraph, please do not change it. Do not put words in **bold** here: for emphasis, use *italic*. Do not use citations here: they are not allowed in the abstract. Footnotes and links are not allowed as well. DO NOT EVER USE ENGLISH SHORT FORMS (i.e., isn't, aren't, don't, etc.). Take a look at the following links about how to write an Abstract:

- `https://writing.wisc.edu/handbook/assignments/writing-an-abstract-for-your-research-paper/`
- `https://www.anu.edu.au/students/academic-skills/research-writing/journal-article-writing/writing-an-abstract`

Search on Google if you need more info.

CHAPTER 1

Generic Chapter

This is a generic chapter of your thesis. Remember to put ANY chapter in a different source file (including introduction and all the others).

For the purpose of this guide, the main LaTeXconstructs and how to use them will be explained here. Other thematic chapters will follow, i.e., which will trace the chapters that should be present in your thesis. Delete this generic chapter once you have learned this contents.

You can write in italic *like this*, you can write in bold **like this**, or you can write using colors like this.

This is an *itemize*, where you can put a list of items, like this:

- item number 1
- item number 2

This is an *enumerate*, where you can put a list of items with numbers, like this:

1. item number 1
2. item number 2

You can cite references like this: [**?**] [**?**], by using the `\cite` directive. You have to copy within `\cite` brackets the label of the entry that you have in the BibTeX file (`.bib`). The `.bib` file of this thesis is `mybib.bib`. he command `\addbibresource` at the top of this main file indicates what BibTeX file you are referring to.

As an example, this is a BibTeX entry:

```
@inproceedings{urias2018cyber,
  title={Cyber Range Infrastructure Limitations and Needs of Tomorrow: A Position Paper},
  author={Urias, Vincent E and Stout, William MS and Van Leeuwen, Brian and Lin, Han},
  booktitle={2018 International Carnahan Conference on Security Technology (ICCST)},
  pages={1--5},
  year={2018},
  organization={IEEE}
}
```

For every online paper that you may read on online libraries, you can download its BibTeX entry. For example:

1. For IEEE Xplore, click on the paper name, then click on “Cite This”, “BibTeX”, and you can find the entry;

2. For Google Scholar, click on the "Cite" voice under the paper name, then click "BibTeX", and you can find the entry.

Just copy and paste such an entry in the .bib file. If you find a paper on Scholar that is nevertheless published by IEEE, by convention you should take the entry from the IEEE website and not from Scholar. To do this, just click on the title of the paper. This will redirect you to the resource page on IEEE Xplore. Once here, follow instructions at point 1.

When you compile, a correct number will automatically be assigned to the citation in the text, and the complete entry will appear at the bottom of the document, in the "Bibliography" chapter.

If you need to cite a generic online resource, which does not necessarily correspond to a scientific paper, use the `@misc` entry in the `.bib` file. A `@misc` entry looks like this:

```
@misc{nist2018,
    author = "{NIST}",
    title = "Cyber Ranges",
    year = "2018",
    howpublished = "\url{https://www.nist.gov/system/files/documents/2018/02/13/cyber_ranges.pdf}",
    note = "[Online; Accessed 2019, 28 November]"
  }
```

You have to manually create this entry from scratch and manually type these fields. Remember not to forget any of these fields. You can choose the label with which to refer to the resource. The title of the website (which you can see at the top of the tab of your browser showing the page) can be used as the title of the resource.

In general, enter a citation of this type for sites only when there are data, phrases, or images that you intend to report. Instead, if you want to cite names of software or hardware devices, prefer the use of the `\footnote`, in which you will only have to specify the URL of the item.

Remember that citations, both in the text and in the image captions, usually go to the end of a sentence, before the fullstop, as in this case [**?**]. In case of long periods, they can also be placed before other detachment signs, such as commas or semicolons, or colons if they precede a list, itemized or enumerated. An exemption is allowed in the event that the name of research projects, described in some scientific resource, is being introduced, as in this case:

Cybertropolis [**?**] is described in a very good paper by Gary Deckard.

Remember to put citations very often to justify your claims, especially when you report data or results. Just consider them as a justification of what you, in an original way, are writing. Citations are not needed to have permission to copy and paste sentences from online resources, which should NEVER be done - always try to rephrase the concept with your words.

This is an image example. Images must ALWAYS be understandable: never introduce images that have text smaller than the text in your document. If you create the images yourself, try not to make them clash too much with the style of your document, and use the same font as this thesis. If they are not images of your own creation, you MUST reference them. In the caption of the image, you need to insert a citation to the resource from which you took the image, at the end of the caption sentence, before the fullstop. Each image you enter MUST be referenced in the text, using a formula similar to this:

Figure 1.1 describes the architecture of the system.

You can refer to the image using `\ref` followed by the image label, that you put in the `\label` entry of the figure. Remember to use the word Figure with a capital F.

Remember that the more your text is adorned with figures, the more understandable, appreciable and readable it becomes.

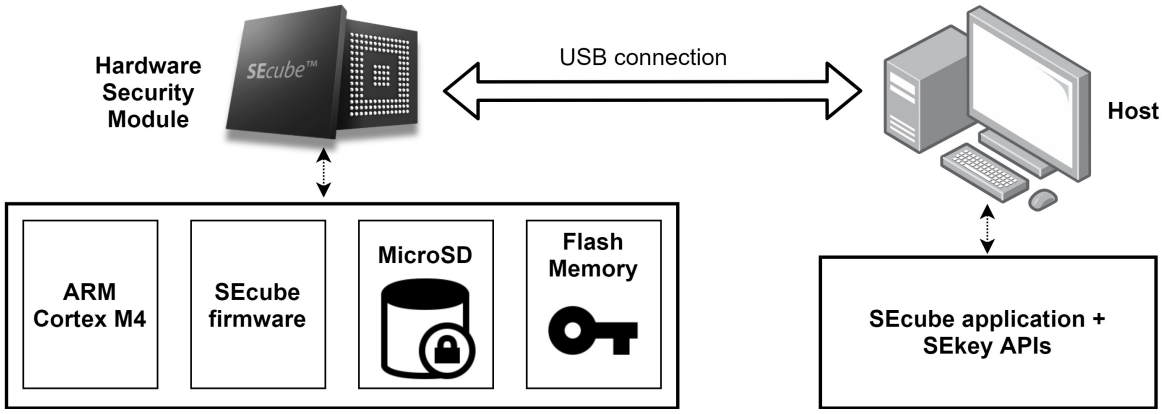

Figure 1.1: This is the image *caption*.

Table 1.1: Preliminary Experimental Results

Benchmark	**Inputs**	**Processing time**
SHA	Message of 100 KB	368449 s
RIJNDAEL	Message of 100 KB	1083568 s
DIJKSTRA	Matrix of 100x100 32-bit integers	324782 s
STRING	1331 50-char strings	178616 s
BITCOUNT	12800 32-bit integers	419545 s

1.1 Section title

This is a section under a chapter. The number of sections also contributes to greater readability of your text, and to a better display of the content in the index. In fact, sections are automatically shown in the Table of Contents. However, try not to make sections shorter than two pages. For smaller portions of your text, use subsections.

You can refer to a section using its label, using the `\ref` directive as for images, like this:

This concept has been explained in Section 1.1.

Remember to use the word Section with a capital S. This is also valid for chapters.

1.1.1 Subsection title

This is a subsection under the section.

The following is a table.

If you want to write a formula, you can do like this:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-ik\frac{2\pi}{N}n} \qquad k = 0, \ldots, N-1 \tag{1.1}$$

Tables and formulas are extensively documented online, and any doubts about their syntax can be easily resolved with a simple search. As for figures and sections, the same rules also apply to tables

and formulas: mandatory reference in the text, possibility to use `\label` to label them, and naming with capital letter (e.g., "as in Table 1.1, as in Formula 1.1).

The following is a piece of code:

```
int func(int N, int M) {
  float (*p)[N][M] = malloc(sizeof *p);
  if (!p)
    return -1;
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
      (*p)[i][j] = i + j;
  print_array(N, M, p);
  free(p);
  return 1;
}
```

You can customize the style of your code, changing the language, the colors of keywords, of comments or the background by changing the settings inside the `\lstset` directive found in the main file. Usually, the listings are not referenced within the text as happens for figures, tables, formulas and sections. Do not overdo the code within your text: use it only for short passages (e.g., function prototypes, or 2 to 5 lines of code within a function to help the reader in better understanding the meaning of the text).

You can also write in-text code using the `\lstinline` directive,
like this: `int main(int argc, char** argv);`.

CHAPTER 2

Introduction

In this first chapter we expect you to introduce the project explaining what the project is about, what is the final goal, what are the topics tackled by the project, etc.
The introduction must not include any low-level detail about the project, avoid sentences written like: we did this, then this, then this, etc.
It is strongly suggested to avoid expressions like 'We think', 'We did', etc...it is better to use impersonal expressions such as: 'It is clear that', 'It is possible that', '... something ... has been implemented/-analyzed/etc.' (instead of 'we did, we implemented, we analyzed').
In the introduction you should give to the reader enough information to understand what is going to be explained in the remainder of the report (basically, expanding some concept you mentioned in the Abstract) without giving away too many information that would make the introduction too long and boring.
Feel free to organize the introduction in multiple sections and subsections, depending on how much content you want to put into this chapter.

Remember that the introduction is needed to make the reader understand what kind of reading he or she will encounter. Be fluent and try not to confuse him or her. The introduction must ALWAYS end with the following formula: The remainder of the document is organized as follows. In Chapter 2, ...; in Chapter 3, ... so that the reader can choose which chapters are worth skipping according to the type of reading he or she has chosen.

CHAPTER 3

Background

In the background chapter you should provide all the information required to acquire a sufficient knowledge to understand other chapters of the report. Suppose the reader is not familiar with the topic; so, for instance, if your project was focused on implementing a VPN, explain what it is and how it works. This chapter is supposed to work kind of like a "State of the Art" chapter of a thesis. Organize the chapter in multiple sections and subsections depending on how much background information you want to include. It does not make any sense to mix background information about several topics, so you can split the topics in multple sections.
Assume that the reader does not know anything about the topics and the tecnologies, so include in this chapter all the relevant information. Despite this, we are not asking you to write 20 pages in this chapter. Half a page, a page, or 2 pages (if you have a lot of information) for each 'topic'(i.e. FreeRTOS, the SEcube, VPNs, Cryptomator, PUFs, Threat Monitoring....thinking about some of the projects...).

CHAPTER 4

Implementation Overview

In this chapter you should provide a general overview of the project, explaining what you have implemented staying at a high-level of abstraction, without going too much into the details. Leave details for the implementation chapter. This chapter can be organized in sections, such as goal of the project, issues to be solved, solution overview, etc.
It is very important to add images, schemes, graphs to explain the original problem and your solution. Pictures are extremely useful to understand complex ideas that might need an entire page to be explained.
Use multiple sections to explain the starting point of your project, the last section is going to be the high-level view of your solution...so take the reader in a short 'journey' to showcase your work.

CHAPTER 5

Implementation Details

5.1 SECube Firmware

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 [3]. The SECube device has been the selected platform that allowed to build the entire Secure Password Manager application on top of it. This is due to, as introduced before, to all the security feature that are intrinsically implemented in the chip itself, both from the point of view of the hardware and software.

In fact, one of the most important security enabling technology is the 3D SiP (System-in-package); the device consists of a number of integrated circuit, each one built on top of the others and enclosed into a single package.

Not only the system is secure from the hardware point of view, that is a necessary condition in order to develop a secure software, but the already present firmware includes some high level functions to generate a secure channel only after the authentication.

Authentication is performed from both parties, in which not only the firmware checks the authenticity of the Host but, the also the Host can do the same. This is possible thanks to a pre-shared key that is stored in the device at the very first initialization of the device itself. A custom C program, after the firmware has been uploaded, allows to setup the master password for the Secure Password Manager. This implies that the password used to authenticate the Host application is configurable only once, allowing to simplify the entire authentication process for the new feature without reducing the security. Further consideration about the user usability and the security aspects are available at Section 5.1.3.

The authentication is performed using a challenge-based authentication from both sides using a MAC (Message Authentication Code) implementation called Hash-based Message Authentication Code (HMAC), that uses a secret and an hash algorithm to verify the solution of the challenge. The challenge is created thanks to the internal TRNG (True Random Noise Generator), that is an hardware included in the chip that generates a true random sequence of bits. Once the challenge has been sent to the party to be authenticated and solved, the solution is read back and checked. If it corresponds to the expected one, the party is authenticated and the channel create. A small possible

weakness study is available at 5.1.3.

5.1.1 Flash memory management

Data are saved in the internal Flash memory, by using a C structure define as follow:

```
typedef struct se3_flash_pass_ {
  uint32_t id;
  uint16_t host_size;
  uint16_t user_size;
  uint16_t pass_size;
  uint8_t* host;
  uint8_t* user;
  uint8_t* pass;
} se3_flash_pass;
```

The Image 5.1 shows a simplified representation of all the used field.

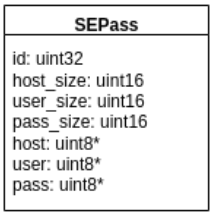

Figure 5.1: SECube firmware password record represented using ER database schema

Internally, some checks has been performed in order to inhibit the user to enter twice the same password record. This check is based on evaluating if the couple hostname and username is already present, and it is performed when the user modifies or creates a new password record. This has been made on purpose, since it has been assumed that a user can have two accounts for the same site and, even if wrong, he or she can use the same password for both of them. This has been implemented using a support method available in `se3_pass.c` file, refer to Code 5.2.

More precisely, data are stored in the flash memory using as few bits as possible to make the implementation able to store a large number of passwords, accordingly to the following table:

Field name	Number of bytes	Description
`id`	4	Id to univocally identify a password, using 32 bits
`host_size`	2	Number of character is the `host`
`user_size`	2	Number of character is the `user`
`pass_size`	2	Number of character is the `pass`
`host`	`host_size`	Hostname of the password record
`user`	`user_size`	Username of the password record
`pass`	`pass_size`	Plain text of the password record

Table 5.1: Flash memory for storing a Password record

The adopted solution allows to use 4 bytes for the id, and 2 bytes for the hostname, username and password length. The length of the other parameters are dependent on the previous three. This implies that at maximum $2^{16} = 65536$ characters can be used, but the system is able to reduce at minimum the occupied size, since the stored bits are not fixed.

Data into the Flash memory are stored as defined by the C structure shown at Table 5.1. Since the internal embedded Flash memory is limited, a possible attack could rely on the fact that creating few record with the hostname, username and password length set to the maximum could fill up the space. This corresponds a DoS (Denial of Service) attack, making the system not more fully usable, by saturating the internal memory.

A solution to this problem could be to limit the number of characters of the hostname, username and password itself to a maximum value. The problem is that even in this case, if the user has an access to the device and it is logged in, he or she will be able to saturate the memory, independently from the dimension of each field. For this reason and for possible future improvements based on using the unused bits in the fields, 2 bytes have been reserved for each field.

5.1.2 Code implementation

The entire solution has been developed using the C language and everything has been built on top of the already present firmware.

Besides all the adaptations to the code that has been necessary to the Secure Password Manager to work correctly, everything is based on two C files under the "*SEcube USBStick Firmware/Project/Src/Device*" directory:

- `se3_sepass.c`
- `se3_pass.c`

A coarse grain classification can be done on the level of data manipulation that the functions inside each one of the two files contains. In the case of `se3_sepass.c`, functions are much more command oriented, allowing to perform rather complex operation by calling directly a function from the just received command. On the other hand, `se3_pass.c` includes the functions for directly managing the Flash memory and abstract over some redundant operations, like the fetch from the storage.

Command dispatcher

The `se3_dispatcher_core.c` file contains the code implementation for managing the custom commands that are necessary to provide to the Host application, in order to manage the passwords.

In order to add the five different commands needed to manage all the Secure Password Manager features (add, delete, modify, search and password generation), a custom command, with id 13 has been added. The five methods have been added by exploiting the sub-command management; part of the command payload is used to identify the id of the method to call. The implementation and management is available at Code 5.1.

```
1  uint16_t sepassword_manager_utilities(uint16_t req_size, const uint8_t* req,
     ↪ uint16_t* resp_size, uint8_t* resp)
2  {
3    uint16_t operation; // the type of operation to be executed
4    memcpy((void*)&(operation), (void*)req, 2);
5    se3_flash_it it = { .addr = NULL};
6    if(!login_struct.y)
7    {
8      return SE3_ERR_ACCESS;
9    }
10
11   se3_flash_it_init(&it);
12   it.addr = NULL;
13   switch (operation)
14   {
```

```
15        case SE3_SEPASS_OP_ADD:
16        return add_new_password(req_size, req+2, resp_size, resp);
17        break;
18        case SE3_SEPASS_OP_MODIFY:
19        return modify_password(req_size, req+2, resp_size, resp);
20        break;
21        case SE3_SEPASS_OP_DELETE:
22        return delete_password(req_size, req+2, resp_size, resp);
23        break;
24        case SE3_SEPASS_OP_GET_BY_ID:
25        return get_password_by_id(req_size, req+2, resp_size, resp);
26        break;
27        case SE3_SEPASS_OP_GETALL:
28        return get_all_passwords(req_size, req+2, resp_size, resp);
29        break;
30        case SE3_SEPASS_OP_GENERATE_RANDOM:
31        return generate_random_password(req_size, req+2, resp_size, resp);
32        break;
33        default:
34        SE3_TRACE(("[sepassword_utilities] invalid operation\n"));
35        return SE3_ERR_PARAMS;
36      }
37      return SE3_OK;
38    }
```

Listing 5.1: "Code for searching if password record is already present"

se3_pass.c

As already introduced before, the `se3_pass.c` contains all the low level operations with the Flash memory. The most important available methods are the following:

- `se3_pass_find`: given a password id, returns true if that id used
- `se3_pass_new`: given a password record, create a new password record in the Flash memory
- `se3_pass_read`: read from the Flash memory the password information of a single password
- `se3_pass_equal`: return true if there is a password with the same hostname and same username. The implementation is available at 5.2

```
1   bool se3_pass_equal(se3_flash_pass* password, se3_flash_it* it)
2   {
3     bool areEquals = false;
4     se3_flash_pass tmp;
5     se3_flash_it_init(it);
6
7     while (se3_flash_it_next(it) && !areEquals)
8     {
9       if (it->type == SE3_TYPE_PASS)
10      {
11        se3_pass_read(it, &tmp);
12
13        if (tmp.id == password->id || (is_str_eq(tmp.host, tmp.host_size,
      ↪ password->host, password->host_size) && is_str_eq(tmp.user, tmp.user_size
      ↪ , password->user, password->user_size)))
14        {
15          areEquals = true;
16        }
17
```

```
18          if(tmp.host != NULL) {free(tmp.host);}
19          if(tmp.user != NULL) {free(tmp.user);}
20          if(tmp.pass != NULL) {free(tmp.pass);}
21        }
22      }
23
24      return areEquals;
25    }
```

Listing 5.2: "Code for searching if password record is already present"

se3_sepass.c

Differently form the `se3_pass.c` file, the `se3_sepass.c` contains high level oriented functions that are directly called by the sub-command command dispatcher for the password manager.

The available methods are the following:

- `add_new_password`: used to generate a new password. The parameters such as the hostname, the username and the password are extracted from the command parameters checked against the current state of the Flash. More precisely, besides the consistency checks are performed before parsing the parameters, the pair hostname and username is searched into the memory, if not present the new password record is created. One important aspect is that the id of the new password is generated outside the device, and it is duty of the Host application to provide a valid value. This has been done in order to increase the flexibility of the solution and to allow the Host to use any kind on enumeration.
- `modify_password`: similarly form the `add_new_password` function, the parameters are read and checked. Only if the id is present, the previous record is deleted and replaced by the new one with all the correct information.
- `delete_password`: this simply deletes the password record by a given id
- `get_all_passwords`: return a list of all the passwords. It is also possible to filter by username or hostname
- `get_password_by_id`: given an id, the password record is returned
- `generate_random_password`: given the length and the set of characters that must be used, a random password is generated using the internal TRNG.

  The password can be generated by using a combination of four different character set:

- Lowercase: abcdefghijklmnopqrstuvwxyz
- Uppercase: ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Number: 1234567890
- Symbol: -_.:;,?&%$!#

The `generate_random_password` has been implemented by exploiting the TRNG that generates twice the number of characters in bytes. This means that if the password to be generated is 100 characters long, the TRNG will be exploited to gather 200 bytes.

This has been done for a specific reason, the first byte is used to identify which character set will be used for the next character of the password. Instead, the second byte is used to select a character

among the selected set.

This solution allows to have that each character set has the same probability to be used so, even if the set sizes are different, a symbol has the same probability to be selected with the respect to an alphabetical one.

This solution has been choose to avoid the problem of a non-uniform distribution of each character type probability. Suppose that all sets are enabled and merged together to form a single set, the probability that taking a random character from the newly generated password is a symbol is not 25% but instead is:

$$\Pr(\text{X} = \text{symbol}) = 100 \cdot \frac{13}{26+26+10+13} = 17.3\%$$

To solve this problem, the first solution has been selected. The Figure 5.2 shows a simplified schema.

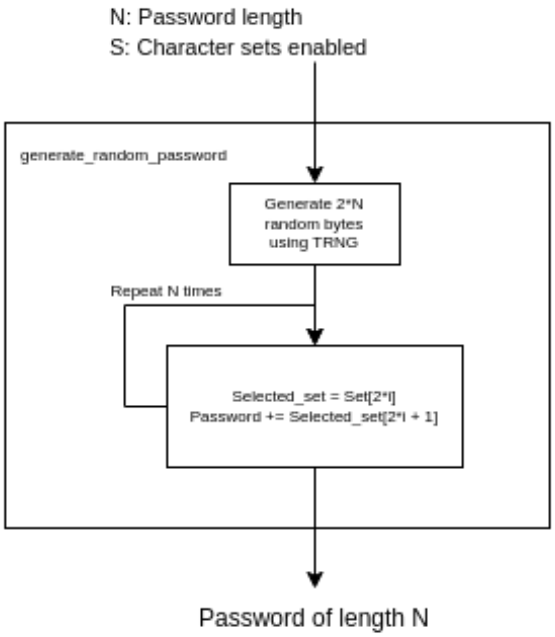

Figure 5.2: SECube firmware password generation

5.1.3 Possible weakness

At the current state-of-the art of the implementation, there is not a way to retrieve or change the password if the used does not know the current one. This is an intrinsic weakness of the solution, since having only one factor authentication that is not supported by a second authentication method, the only way to gain access to the stored password would have been creating a support method that would have reduced the security. This generates a problem: giving the possibility to the user to retrieve the master password in some way, would have reduced the overall security. For this reason, even if the user usability could be reduced, the main focus was to not disclose the password to who is not able to be authenticated.

From the point of view of the HMAC used to perform the Host and Device mutual authentication the security is intrinsically provided by the fact that the secret is stored in a secure hardware. The

fact that also the Host application has the ability to verify the Device using the same challenge-based authentication explained before, implies that the solution generated by the firmware has been generated by using the same shared key that is used to authenticate the Host application. The following sentences explain how an bruteforce attack could be used against this type of authentication, with focus on a particular use case.

From the Chrome Extension usability definition, the user has to enter the password in order to unlock the extension itself and be allowed to manage the password. If an attacker is doing *piggyback*, he or she will see only the number of inserted character (since characters will be replaces with black dots) or some pressed keys. The problem with this is that, if the firmware challenge has been generate and available to the attacker in some way (like stealing the SECube for some minutes), he or she can perform a bruteforce attack. The time will be drastically reduced if the number of character is known. One solution to solve this would be to not allow the Host application to authenticate the SECube itself, but this is not secure for obvious reasons. During the early development of this project, it was proposed to implement a delayed authentication; if the user performs X wrong login attempts, the next one must be done after Y seconds, otherwise an error is always generated. The fact that challenge is independently generated from the login API, makes also this solution unfeasible. The feasibility of the attack strictly depends on the importance of the stored passwords and it's a matter of effort/cost, since the high security level of the challenge used.

5.1.4 Flash the firmware

This tutorial needs a working version of Eclipse for C/C++ and the AC6 Tools are properly installed in order to build the firmware and flash it to the SECube device. The configuration describes uses the following configuration:

- ST-Link/v2 programmer
- Eclipse IDE for C/C++ Developers (includes Incubating components) Version: 2022-03 (4.23.0) Build id: 20220310-1457
- Ubuntu 18.04 5.4.0-117-generic

Hardware and Compiler setup

In this Section are reported the instructions you need to follow to properly connect the SEcube DevKit to the Host PC and to Programmer/debugger, refer to Figure 5.3. A more detailed configuration tutorial is available via the official SECube Wiki.

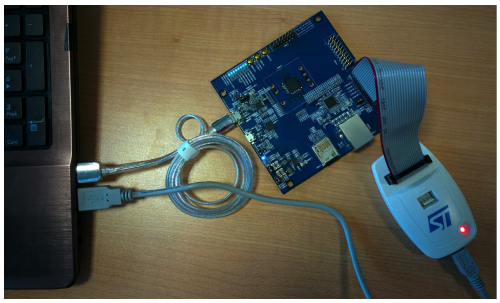

Figure 5.3: Connection between the STLink/v2 programmer and the SEcube DevKit

Assembling is composed of the following two steps in order to obtain the situation that is available at Figure 5.4:

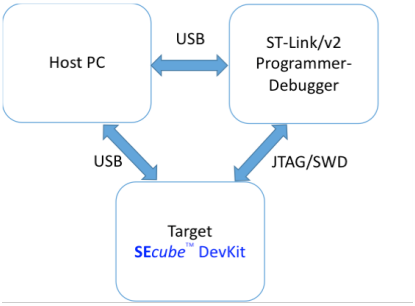

Figure 5.4: System Architecture

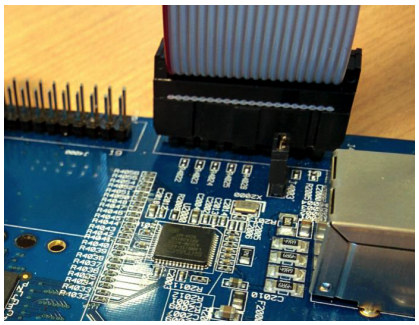

Figure 5.5: Connection between the STLink/v2 programmer and the SEcubeTM DevKit, close-up (highlighted in red) on the JTAG connector orientation

1. Connect the SEcube DevKit with the programmer by means of the JTAG/SWD cable: the cable should be inserted on the JTAG docks on both the programmer (in this case the orientation of the plug is forced from the dock) and the DevKit (in this case you must pay attention in inserting the plug on top of both lines of connectors and with its protrusion oriented towards the inner side of the DevKit).

2. Connect the ST-Link/v2 with the PC by means of the USB cable.

The system assembled is shown in Figure 5.3, while a close-up on the JTAG connection is in Figure 5.5.

In order to be able to build and flash the firmware, the AC6 Tools must be installed via Eclipse. The AC6 Tool will install the GNU Embedded Toolchain for ARM, which is a ready-to-use, open source suite of tools for C, C++ and Assembly programming targeting ARM Cortex-M and Cortex-R family of processors. It includes the GNU Compiler (GCC) and is available free of charge directly from ARM for embedded so ware development on both Windows and Linux operating systems. The reference platform for this document is the System Workbench for STM32 (SW4STM32) Eclipse plugin.

SW4STM32 is an integrated environment that includes:

- Building tools (GCC-based ARM cross compiler, assembler and linker);
- OpenOCD and GDB debugging tools;
- Flash programming tools

To install SW4STM32 as an Eclipse plugin:

1. launch Eclipse IDE

2. on the toolbar, click Help Install New Software...
3. in the Available Software window, click Add
4. in the Add Repository window, set Name and Location fields as follows, and then click OK:
   - Name: System Workbench for STM32 - Bare Machine edition
   - Locaton: http://www.ac6-tools.com/Eclipse-updates/org.openstm32.system-workbench.site
5. select OpenSTM32 Tools and click Next
6. accept the license agreement and click Finish to start the plugin installation, continue the installation also if a warning for incompatible or unsigned components is prompted
7. restart Eclipse

Firmware flashing

Once the SECube has been connected to the Host computer via both the ST-Link/v2 programmer and the USB connection, the firmware can be imported, compiled and flashed.

The latest firmware version has been already compiled and available at "*SEcube USBStick Firmware/Project/Eclipse/USBStick/Release/USBStick.elf*" but it is possible to import the project into Eclipse and recompile it. If you want to use the pre-compiled version, you can skip the following sections and go to Section 5.1.4.

In order to import the firmware and the software for performing the init, you need to click File and then Import..., as in Figure 5.6. At this point after having selected Existing Projects into Workspace (Figure 5.7), the first two projects must be imported (Figure 5.8).

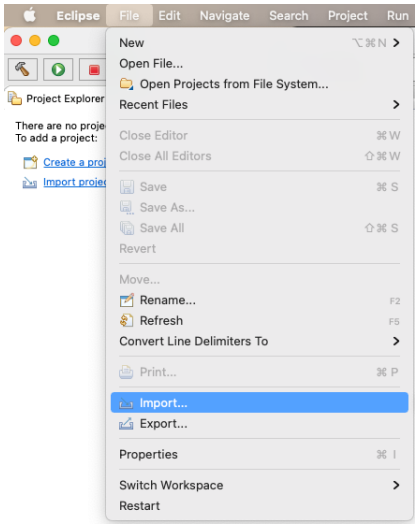

Figure 5.6: Project import in Eclipse

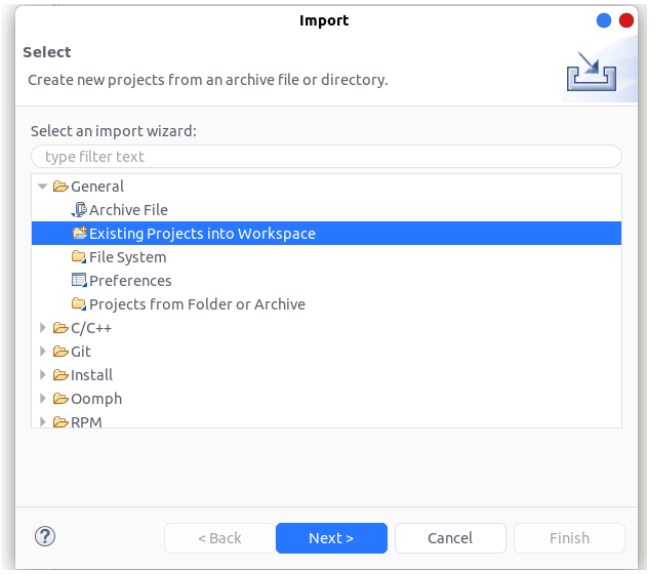

Figure 5.7: Import of projects

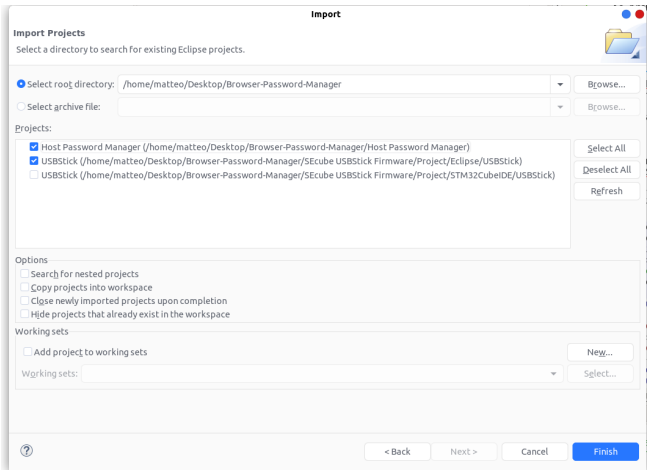

Figure 5.8: USBStick firmware and Host Password Manager for Initilization

The first project will be used during configuration of the device while the second one is the firmware itself.

At this point, on the right you should have two projects, you have to Right click over the “USB-Stick” one and select “Build Project” (refer to Figure 5.9).

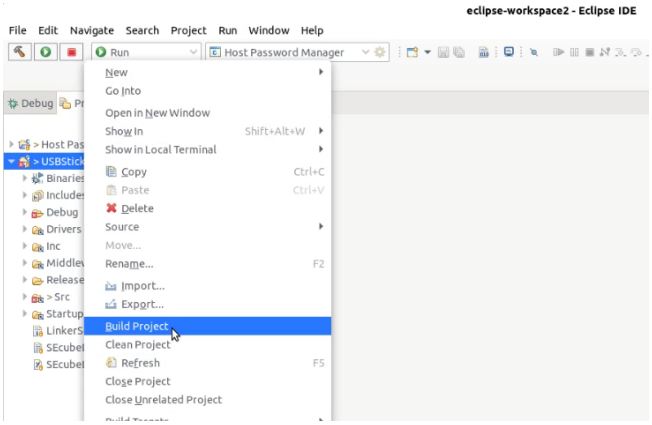

Figure 5.9: Build the firmware

Configuring the device

Once the firmware, you have to first of all to erase the chip in order to remove the previous pin configuration, by doing a Right click on the project and then “Target” and then “Erase Chip...”. Once it has finished, you can flash the firmware into to the device by clicking “Target” and then “Program Chip...” (refer to Figure 5.10). In the next window you have to select the “Release” version and flag the “Reset after program” before clicking “OK”.

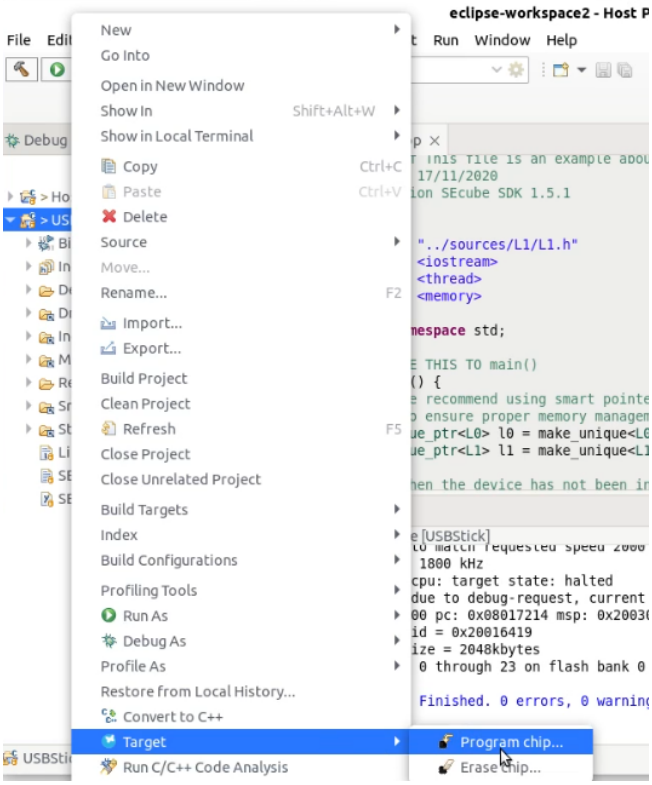

Figure 5.10: Flash the firmware

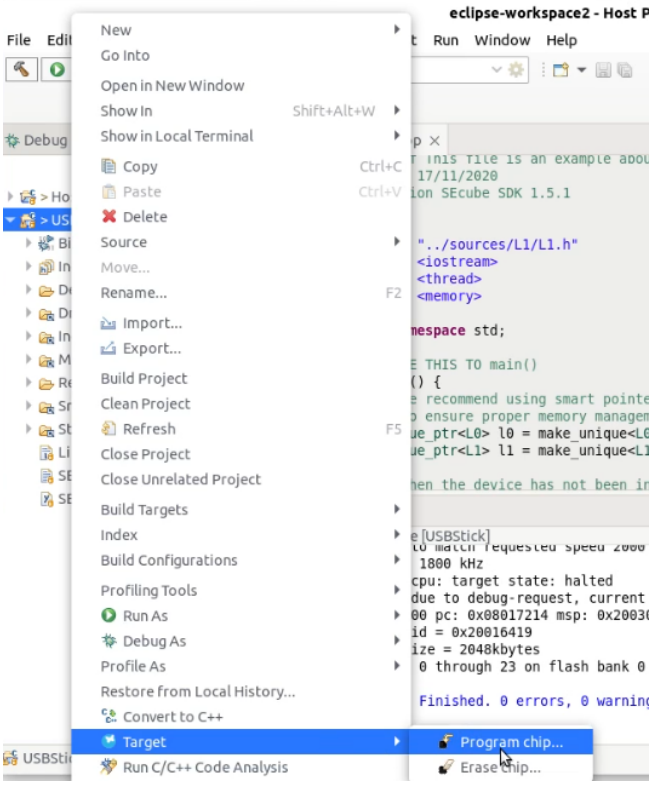

Figure 5.11: Firmware Release version selection

At this point, once the firmware has been flash, you need to configure the device by setting a pin. Now it's the turn of the second project called "Host Password Manager".

You have to open the `device_init.cpp` file and check that the name of the first method is set to `main`. At this point, you can perform the compilation and run the program. This allows to initialize the device and set the master password for the Secure Password Manager (at line 49 it's possible to change the pin). From now, the device is ready to receive commands by the Host Middleware application in order to manage all passwords features.

CHAPTER 6

Results

In this chapter we expect you to list and explain all the results that you have achieved. Pictures can be useful to explain the results. Think about this chapter as something similar to the demo of the oral presentation. You can also include pictures about use-cases (you can also decide to add use cases to the high level overview chapter).

6.1 Known Issues

If there is any known issue, limitation, error, problem, etc...explain it in this section. Use a specific subsection for each known issue. Issues can be related to many things, including design issues.

6.2 Future Work

Adding a section about how to improve the project is not mandatory but it is useful to show that you actually understood the topics of the project and have ideas for improvements.

CHAPTER 7

Conclusions

This final chapter is used to recap what you did in the project. No detail, just a high-level summary of your project (1 page or a bit less is usually enough, but it depends on the specific project).

Bibliography

[1] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[2] Leslie Lamport (1994) *LATEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.

[3] What is SEcube, `https://www.secube.blu5group.com/`.

APPENDIX A

User Manual

In the user manual you should explain, step-by-step, how to reproduce the demo that you showed in the oral presentation or the results you mentioned in the previous chapters.
If it is necessary to install some toolchain that is already well described in the original documentation (i.e., Espressif's toolchain for ESP32 boards or the SEcube toolchain) just insert a reference to the original documentation (and remember to clearly specify which version of the original documentation must be used). There is no need to copy and paste step-by-step guides that are already well-written and available.
The user manual must explain how to re-create what you did in the project, no matter if it is low-level code (i..e VHDL on SEcube's FPGA), high-level code (i.e., a GUI) or something more heterogeneous (i.e. a bunch of ESP32 or Raspberry Pi communicating among them and interacting with other devices).

APPENDIX B

API

If you developed some source code that is supposed to be used by other software in order to perform some action, it is very likely that you have implemented an API. Use this appendix to describe each function of the API (prototype, parameters, returned values, purpose of the function, etc).