



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

1	Generic Chapter	2
1.1	Section title	4
1.1.1	Subsection title	4
2	Introduction	6
3	Background	7
4	Implementation Overview	8
5	Implementation Details	9
5.1	SECube Firmware	9
5.1.1	Flash memory management	10
5.1.2	Code implementation	11
5.1.3	Possible weakness	14
5.2	Host Middleware	15
5.2.1	The web server	15
5.2.2	The REST APIs	15
5.2.3	Session Management	15
5.2.4	Timestamp and Timeout Management	16
5.2.5	How to interface Python with C++?	16
5.2.6	How to secure Python code?	17
6	Results	19
6.1	Known Issues	19
6.2	Future Work	19
7	Conclusions	20
A	User Manual	22
A.0.1	Firmware	22
A.1	Host Middleware	28
A.1.1	Linux	28
A.1.2	Windows	29
B	API	32
B.0.1	/api/v0/time	32
B.0.2	/api/v0/devices	32
B.0.3	/api/v0/device/{id}/sessions	32
B.0.4	/api/v0/device/{id}/generate	32

B.0.5	/api/v0/device/{id}/passwords	33
B.0.6	/api/v0/device/{id}/password/{id}	33

List of Figures

1.1	This is the image <i>caption</i>	4
5.1	SECube firmware password record represented using ER database schema	10
A.2	System Architecture	23
A.3	Connection between the STLink/v2 programmer and the SEcubeTM DevKit, close-up (highlighted in red) on the JTAG connector orientation	23
A.1	Connection between the STLink/v2 programmer and the SEcube DevKit	23
A.4	Project import in Eclipse	25
A.5	Import of projects	25
A.6	USBStick firmware and Host Password Manager for Initilization	26
A.7	Build the firmware	26
A.8	Flash the firmware	27
A.9	Firmware Release version selection	27

List of Tables

1.1	Preliminary Experimental Results	4
5.1	Flash memory for storing a Password record	10

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 L^AT_EXsource 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 L^AT_EX constructs 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`. The 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.

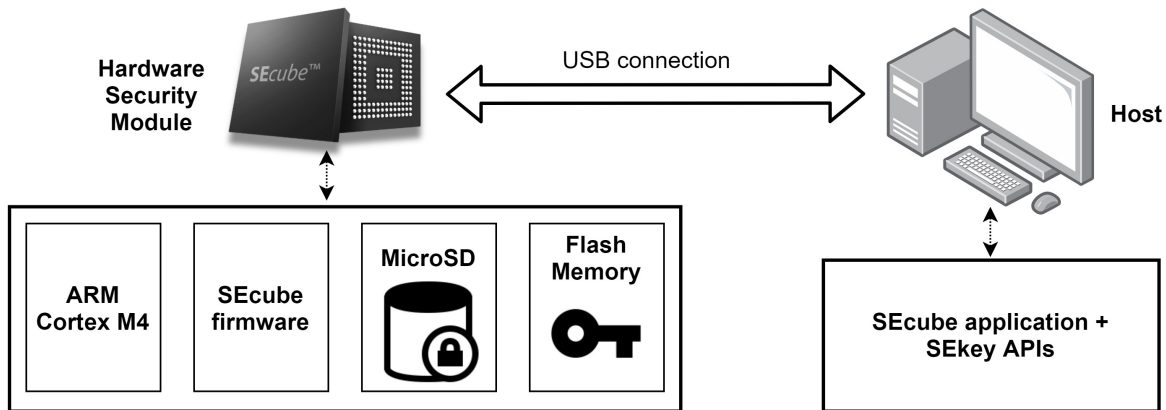
Figure 1.1: This is the image *caption*.

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 inte- gers	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:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-ik \frac{2\pi}{N} n} \quad k = 0, \dots, N-1 \quad (1.1)$$

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:

```
1 int func(int N, int M) {  
2     float (*p)[N][M] = malloc(sizeof *p);  
3     if (!p)  
4         return -1;  
5     for (int i = 0; i < N; i++)  
6         for (int j = 0; j < M; j++)  
7             (*p)[i][j] = i + j;  
8     print_array(N, M, p);  
9     free(p);  
10    return 1;  
11 }
```

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 multiple sections.

Assume that the reader does not know anything about the topics and the technologies, 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:

```

1  typedef struct se3_flash_pass_ {
2      uint32_t id;
3      uint16_t host_size;
4      uint16_t user_size;
5      uint16_t pass_size;
6      uint8_t* host;
7      uint8_t* user;
8      uint8_t* pass;
9  } se3_flash_pass;

```

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

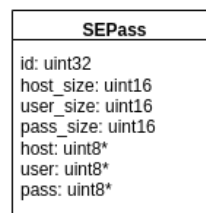


Figure 5.1: SECube firmware password record represented using ER database schema

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
<code>id</code>	4	Id to univocally identify a password, using 32 bits
<code>host_size</code>	2	Number of character is the <code>host</code>
<code>user_size</code>	2	Number of character is the <code>user</code>
<code>pass_size</code>	2	Number of character is the <code>pass</code>
<code>host</code>	<code>host_size</code>	Hostname of the password record
<code>user</code>	<code>user_size</code>	Username of the password record
<code>pass</code>	<code>pass_size</code>	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,
2  ↪ uint16_t* resp_size, uint8_t* resp)
3  {
4  uint16_t operation; // the type of operation to be executed
5  memcpy((void*)&(operation), (void*)req, 2);
6  se3_flash_it it = { .addr = NULL};
7  if (!login_struct.y)
8  {
9  return SE3_ERR_ACCESS;
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,
14            ↪ password->host, password->host_size) && is_str_eq(tmp.user, tmp.user_size
15            ↪ , password->user, password->user_size)))
16            {
17                areEquals = true;
18            }
19        }
20    }
21 }

```

```
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 from 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 from 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 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 100 bytes.

This has been done for a specific reason, each byte is used to select a character from the set that has been generated from the union of all enabled sets.

This solution allows to have that each character in the cumulative set has the same probability to be used.

This solution has been choose to avoid the problem of a non-uniform distribution of each character probability. If all sets are enabled and merged together to form a single set, as in this implementation, the probability that taking a random character from the newly generated password is exactly the ‘A’ one is:

$$\Pr(X = A) = 100 \cdot \frac{1}{26+26+10+13} = 1.3\%$$

The firmware function used to generate a random password is based on randomly select N characters from the enabled sets (alphabet lowercase and uppercase, numbers and symbols). The current implementation, is totally random but is able to ensure that at least one character from all the enabled sets is selected. This has been implemented by performing a loop over the just generated password, if it does not contains at least one character for each enabled set, the computation is repeated. In order to avoid an infinite loop, the upper limit of the number of iterations has been set to 100.

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 is a matter of effort/cost, since the high security level of the challenge used.

5.2 Host Middleware

The Host Middleware is a software intended to run in the user's PC (for example as a daemon on Linux or as a service on Windows) and to provide a secure connection between the user's PC and the SECube board. This means that it acts as a bridge between the Chromium Extension (the frontend for the user) and the board, thus the Middleware is developed with security in mind.

It's main job is to serve some HTTPs requests. In fact, it provides REST APIs to allow the Chromium Extension to interact with the features exposed by the SECube's firmware. This means that it acts as a web server with HTTPs support in order to provide a secure connections.

5.2.1 The web server

HTTPs is a secure protocol that uses a TLS connection to provide a secure connection between two endpoints, and it's a replacement for the HTTP protocol. This means that HTTPs provides the following benefits:

- Authentication
- Privacy: the connection is encrypted and the data is encrypted
- Integrity: the data is signed and the signature is verified

Thus, HTTPs helps to avoid the risk of eavesdropping, which is a risk that can be exploited by an attacker to intercept the data and modify it. More in general, it avoids *Man In The Middle* attacks.

The middleware is developed mainly in Python. To implement the web server, Flask is used as module. It natively supports HTTPs and a self-signed certificate is used, generated with *openssl*.

5.2.2 The REST APIs

The exposed APIs are totally compliant to the REST principles. It uses cookies to authenticate the user, and it uses JSON as exchange data format.

The main API is the one that allows to create a session. Once a session is created (via a successful authentication), a cookie is generated and sent to the browser. The browser will then store securely the cookie and the extension will automatically attach it to each request. In the end, the cookie is strictly needed to interact with all other APIs.

The endpoint to create a session is `POST /api/v0/device/0/session?pin=<pin>&endtime=<endt>`. The `pin` parameter is the PIN code of the board, used to unlock it, while the `endtime` parameter is the time in seconds until the session expires. When the session expires, the middleware will automatically invalidate it and each subsequent request will result in `403 Unauthorized`. The `endtime` parameter is a timestamp in seconds, and it's relative to the middleware's one. The middleware is capable of generating it internally and the current timestamp can be obtained via `GET /api/v0/time`. For a more complete description of the API, see the appendix B.

5.2.3 Session Management

When a cookie is created, it contains only the session ID. This ID is used to identify the session, and it's used to identify the user. The session ID is generated by the middleware and it's unique for each session. This means that on the user's side, only the ID is stored instead of any other sensitive

information. The user is protected by *client identity steal* attacks thanks to the security given by the browser in storing it locally.

On the middleware side, a session corresponds to a file stored in the file system, in the same directory as the middleware's executable. The file name is the session ID. The file contains the following information:

- The board's PIN given by the user when the session was created
- The endtime of the session (timestamp in seconds)

In order to avoid possible attacks because of the files stored in clear in the file system, the session is encrypted with a key that is generated by the middleware. At each startup of the middleware, a 2048 bit RSA key is generated and stored in the RAM, and each previously created session is invalidated and destroyed. The public/private keys are used to encrypt/decrypt on the fly the requested session file. The encryption/decryption is done on-the-fly and then the session is stored in the file system, so a non encrypted version of the file will never appear in the file system. Both the PIN and the endtime are encrypted, along with other side informations.

5.2.4 Timestamp and Timeout Management

In order to avoid the risk of *time-leap* or *time travel* attacks, the middleware uses an internal timestamp instead of the PC's one. So, even if a malicious user changes the PC's time, the middleware's timestamp will continue to update itself correctly and sessions will expire correctly.

In order to generate the timestamp, the middleware uses a dedicated thread that periodically (each seconds) updates the timestamp. The same information can be accessed via `GET /api/v0/time`. The timestamp is updated each second, so it's not possible to get a timestamp that is in the past.

At each request, the middleware:

1. finds the session file corresponding to the session ID in the request (associated to the cookie sent by the Extension)
2. decrypts the file
3. gets the stored endtime timestamp
4. gets the current timestamp (so the middleware's timestamp)
5. compares the endtime with the current timestamp and if the endtime is reached (greater or equal), the session is invalidated and the request is replied with `403 Unauthorized`
6. gets the stored PIN
7. tries to authenticate the user with the PIN
8. if the *login* is unsuccessful, the request is replied with `403 Unauthorized`, otherwise the request continues with the operation requested by the user

5.2.5 How to interface Python with C++?

As mentioned above, the middleware is developed in Python. However, the HOST libraries used to communicate with the board are written in C++. This means that the Python code needs to be able to interface with the C++ libraries. In order to do that, the middleware uses the *Ctypes* module. It's

a builtin module that allows to interface with C libraries.

The actual implementation sees some wrappers (both for the *L0* and *L1* libraries). Those wrappers are C functions that uses only C's primitive types as arguments and as return values, because of the way ctypes works. An example of function wrapper is the following one, used to initialize the *L0* library:

```

1 #ifdef _WIN32
2 #include <Windows.h>
3 #define __MODIFIER __declspec(dllexport)
4 #else
5 #define __MODIFIER
6 #endif
7
8 #define EXPORT_FUNC(_type, _name) extern "C" __MODIFIER _type _name
9
10 EXPORT_FUNC(void *, createL0Instance)() {
11     return new(std::nothrow) L0;
12 }

```

Once C wrappers are all defined and implemented, all the code needs to be compiled. In this case there is no *main* function, so there is no an executable file to run. Because those functions are meant to be called from the external (i.e. the Python code), the code is compiled as a shared library. On linux, they are compiled as *.so* files. On Windows, they are compiled as *.dll* files. The shared library is then loaded by the Python code.

With ctypes, the function wrapper is easily invoked. The first thing that is done is to tell to ctypes what are the arguments and return types of the function, then the function can be called. Because this procedure must be done for each function, a python class has been created to encapsulate the functions. There is a class both for the *L0* and *L1* libraries, and this is an example:

```

1 class L0:
2     def __init__(self, path_lib: str):
3         self._libname = path_lib
4         self._c_lib = ctypes.CDLL(self._libname)
5
6         # Create L0 instance
7         self._c_lib.createL0Instance.argtypes = []
8         self._c_lib.createL0Instance.restype = ctypes.c_void_p
9
10        self._l0inst = self._c_lib.createL0Instance()

```

5.2.6 How to secure Python code?

The python code is not secure. While the C code is compiled into executable code and it's a very difficult job to reverse engineer, the Python code is directly interpreted on the fly, so there is no compilation and the code is in clear. Moreover, it can suffer from *code injection* attacks, so the code must be protected against those attacks.

In order to achieve this, a python library is used, called *pyarmor*. Its first job is to obfuscate the code, so write the code in such a way it can't be understood easily by an human. This protects against *code injection* and reverse engineering. The resulting obfuscated script is something like this:

```

1 from pytransform import pyarmor_runtime
2 pyarmor_runtime()
3 __pyarmor__((__name__, __file__, b'\x06\x0f...'))

```

Moreover, the original python code (that now is obfuscated) is wrapped with functions that insert timers to prevent debugging and to prevent old stack traces dump. This is done by letting the python code run normally and at the end of each function call (where each function call creates a new stack frame), the frame is cleared.

Finally, the last precaution is to wrap everything into a single executable. The middleware is mainly composed of different python files, the shared library and the private and public keys. Using a library called *pyinstaller*, the middleware is packed into a single executable. What *pyinstaller* does is to create an executable that when is executed it creates a temporary folder and extract from itself all the files together with a standalone version of the python interpreter. This is done so that the middleware can be executed without the need to install the python interpreter and without the library installed. Moreover, it protects by possible *code injection* attacks done at the interpreter level. The final job of *pyinstaller* is to encrypt the content of the executable. Because the executable needs to know the key to decrypt it when it's requested by the user, the key is hardcoded into the executable itself, so it's relatively easy to obtain the key but a malicious user needs to do some reverse engineering of the executable itself so it's an increase of time spent on trying to attack the middleware.

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

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 T_EX Book*, Addison-Wesley Professional.
- [2] Leslie Lamport (1994) *L^AT_EX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.
- [3] What is SEcube, <https://www.secube.blugroup.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).

A.0.1 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 A.1. A more detailed configuration tutorial is available via the official SECube Wiki.

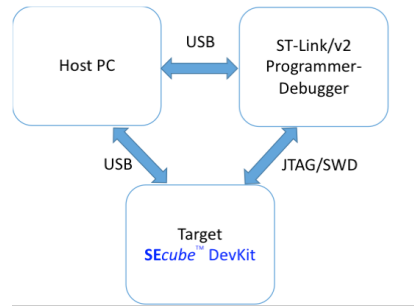


Figure A.2: System Architecture

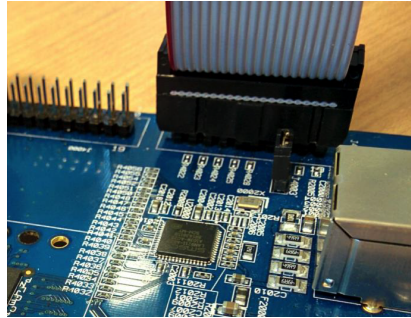


Figure A.3: Connection between the STLink/v2 programmer and the SEcube™ DevKit, close-up (highlighted in red) on the JTAG connector orientation

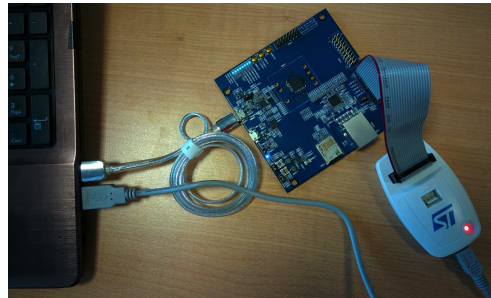


Figure A.1: Connection between the STLink/v2 programmer and the SEcube DevKit

Assembling is composed of the following two steps in order to obtain the situation that is available at Figure A.2:

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 A.1, while a close-up on the JTAG connection is in Figure A.3.

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 software 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
 - Location: <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 A.0.1.

In order to import the firmware and the software for performing the init, you need to click File and then Import..., as in Figure A.4. At this point after having selected Existing Projects into Workspace (Figure A.5), the first two projects must be imported (Figure A.6).

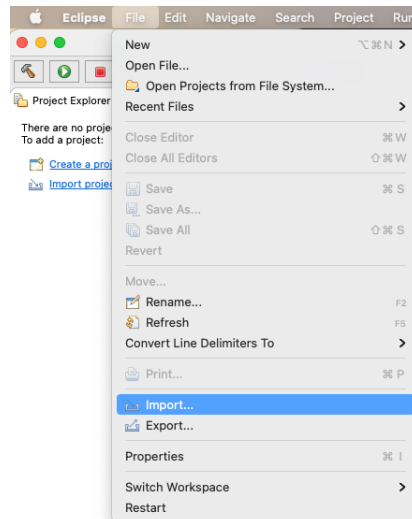


Figure A.4: Project import in Eclipse

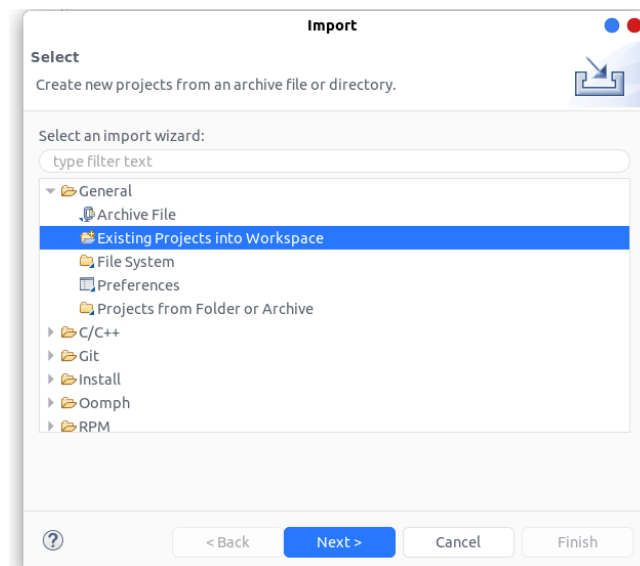


Figure A.5: Import of projects

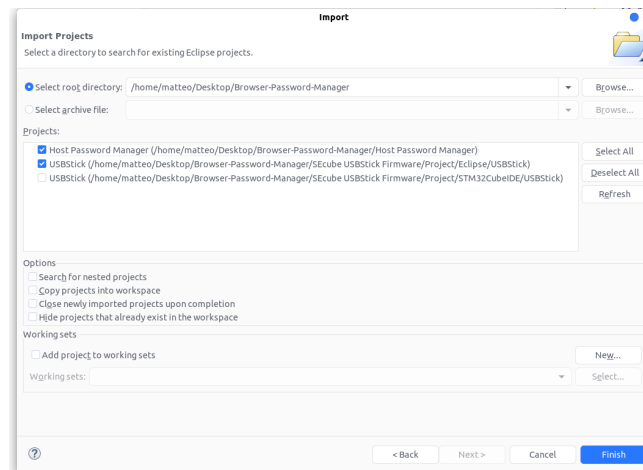


Figure A.6: USBStick firmware and Host Password Manager for Initilization

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 A.7).

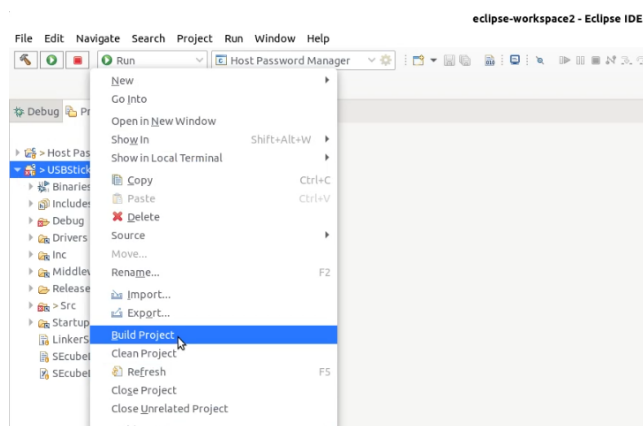


Figure A.7: Build the firmware

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 A.8). In the next window you have to select the “Release” version and flag the “Reset after program” before clicking “OK”.

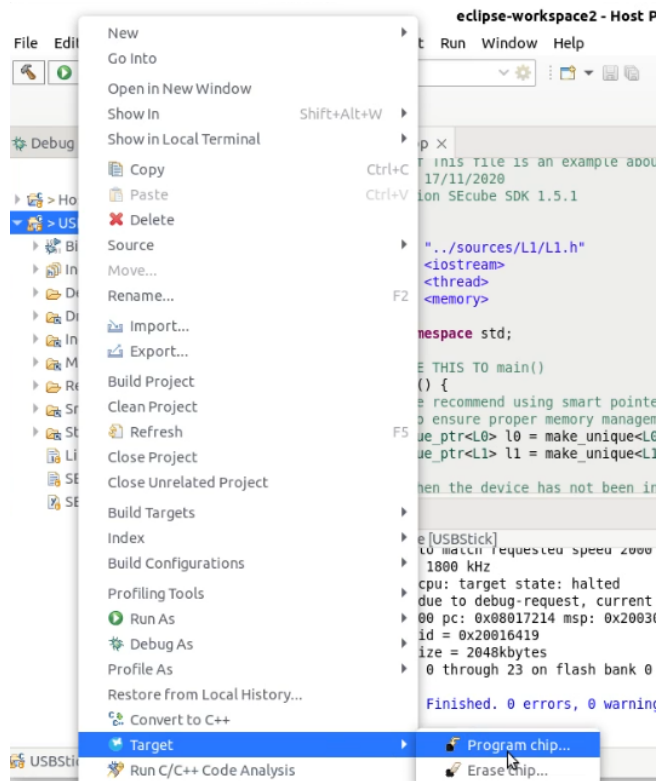


Figure A.8: Flash the firmware

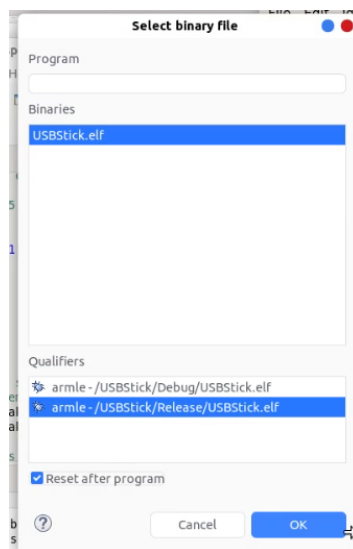


Figure A.9: Firmware Release version selection

At this point, once the firmware has been flash, you need to configure the device by setting a pin. Now it is 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 is 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.

A.1 Host Middleware

In this section, we will describe how to build and run the host middleware, both on Linux and Windows. The build process is not necessary because a ready-to-run executable will be provided both for Linux and Windows. However, if there are problems in executing them, the build process can be used to try to launch the executable.

☞ Because of all the dependencies and operations to do to build, some problems may occur. This section will try to indicate all the necessary software that is required, but unfortunately the successful build of the host middleware is not guaranteed because of the heterogeneous nature of our computers.

A.1.1 Linux

Luckily, on Linux the build process is pretty straightforward. The only thing that needs to be done is to install the necessary software. The following is a list of software that is required to build the host middleware.

- **Python 3.9.x** (tested with 3.9.7). Check that your PATH environment variable points to the Python executable *python3*.
- **pip 20.3.4** (tested with 20.3.4). Check that your PATH environment variable points to the pip executable *pip* and refers to the correct Python version.
- **gcc 11.x** tested with 11.2.0. Check that your PATH environment variable points to the gcc executable *gcc*.
- **g++ 11.x** tested with 11.2.0. Check that your PATH environment variable points to the g++ executable *g++*.
- **GNU Make** tested with GNU Make 4.3. Check that your PATH environment variable points to the GNU Make executable *make*.
- **git** Check that your PATH environment variable points to the git executable *git*.

Here a few steps to build the host middleware if all the required software is installed correctly. Note, it can change depending on the used linux distribution. It may require further steps to install the dependencies.

```
1 $ git clone https://github.com/SEcube-Project/Browser-Password-Manager.git
2 $ cd Browser-Password-Manager/HostMiddleware
3
4 # To build the shared library
5 $ make clean
6 $ make -j4 lib.so
7
8 # To run the script as-is (include build of lib.so)
9 $ make clean
```

```
10 $ make -j4 run
11
12 # To compile, obfuscate and pack into a single executable
13 $ make clean
14 $ make -j4 dist
15 $ ./BPMMiddleware
```

Listing A.1: bash version


A.1.2 Windows

Unfortunately, on Windows there is a lot more work to do.

Python

Python 3.9.x is needed. If you are not sure it's installed on your system, try to launch a Powershell console and type `python --version`. If you get `Python 3.9.0` or similar means that Python is installed. Otherwise, you need to install it.

👉 **ATTENTION:** if the Windows Store opens up, close it! You need to install it in the *classic way* otherwise strange things will happen later on. In the same way, if you installed python from the Windows Store, uninstall it and download it from the official Python webpage.

 **ATTENTION:** if python is not available from the Powershell after manual installation, try to reboot. If it's still not available, you need to manually specify the Python's executable path. Start menu, type Python, right-click and select "Open File location". Most likely it will head you to the Start Menu Shortcuts, so right-click again on the Python 3.9 folder and click on "Open file location". Select the path and copy in the clipboard.

In the start menu, search for "environment" and click "Edit the system environment variable". Click on "environment variables" button, select "Path", click "Edit", click "New" and paste the path you previously copied.

Confirm and close everything, the Powershell too. Open it again and check if now python is available.

C++ Compiler - Buildtools

Head to the start menu and look for x64 Native Tools Command Prompt for VS 2022 (if you are on a 32 bit system, look for x64 Native Tools Command Prompt for VS 2022). Open it, and a terminal emulator will show up. Type `cl`. If you get *'cl' is not recognized as an internal or external program...* means that something is missing, otherwise you will get the following message and it means that the compiler is installed. Same reasoning must undergo with the `link` command.

```

1 C:\Program Files\Microsoft Visual Studio\2022\Community>cl
2 Microsoft (R) C/C++ Optimizing Compiler Version 19.32.31329 for x64
3 Copyright (C) Microsoft Corporation. All rights reserved.
4
5 usage: cl [ option... ] filename... [ /link linkoption... ]
6
7 C:\Program Files\Microsoft Visual Studio\2022\Community>link
8 Microsoft (R) Incremental Linker Version 14.32.31329.0
9 Copyright (C) Microsoft Corporation. All rights reserved.
10
11 usage: LINK [options] [files] [@commandfile]
12
13 options:
14
15     /ALIGN:#
16     /ALLOWBIND[:NO]
17     /ALLOWISOLATION[:NO]
18     /APPCONTAINER[:NO]
19     /ASSEMBLYDEBUG[:DISABLE]
20     /ASSEMBLYLINKRESOURCE:filename
21     /ASSEMBLYMODULE:filename
22     /ASSEMBLYRESOURCE:filename[, [name][, PRIVATE]]
23     /BASE:{address[, size] | @filename, key}
24     /CLRIMAGETYPE:{I JW | PURE | SAFE | SAFE32BITPREFERRED}
25     /CLRLOADEROPTIMIZATION:{MD | MDH | NONE | SD}
26     /CLRSUPPORTLASTERROR[:{NO | SYSTEMDLL}]
27     /CLRTHREADATTRIBUTE:{MTA | NONE | STA}
28     /CLRUNMANAGEDCODECHECK[:NO]
29     /DEBUG[:{FASTLINK | FULL | NONE}]
30     /DEF:filename

```

```
31 /DEFAULTLIB:library
32 /DELAY:{NOBIND|UNLOAD}
33 /DELAYLOAD:dll
34 /DELAYSIGN[:NO]
35 /DEPENDENTLOADFLAG:flag
36 /DLL
37 /DRIVER[:{UPONLY|WDM}]
38 /DYNAMICBASE[:NO]
39 /EMITVOLATILEMETADATA[:NO]
40 (press <return> to continue)
```

Listing A.2: bash version

If something is missing (or the Visual Studio's Command Prompt Tool is not available), Visual Studio must be installed. Go to <https://visualstudio.microsoft.com/downloads/> and download Visual Studio Community edition. Once the installer is downloaded, launch it, select *Visual Studio Community 2022* (click on Modify if Visual Studio is already installed) and select *Desktop Development with C++*. The following parts must be installed:

- MSVC v143 - VS 2022 C++ x64/x86 build Tools
- Windows 10 SDK
- C++/CLI support for v143 build Tools
- C++ Modules for v143 build tools
- C++ Clang tools for Windows

Repeat from the beginning, be sure that the *Visual Studio's Command Prompt* is installed and the compiler is available.

How to build

Now everything should be installed. Open the Visual Studio's Command Prompt, head to the Host-Middleware folder (with the CD command) and type `compile_win.bat`. It will compile everything and pack into a single `BPMMiddleware.exe` executable that eventually you can run, if everything went fine.

APPENDIX B

API

Middleware HTTPs' API

B.0.1 `/api/v0/time`

Used to work with the timestamp. The timestamp is an integer in seconds. The supported HTTP methods are:

- **GET**: returns the current timestamp.

B.0.2 `/api/v0/devices`

Used to work with the devices. It allows to obtain all the connected boards, in particular for each device the ID, Name and Serial are returned. The API is currently not used by the Extension because it's supposed that only one device at a time is connected. The supported HTTP methods are:

- **GET**: returns the list of devices.

B.0.3 `/api/v0/device/{id}/sessions`

Used to manage sessions. The supported HTTP methods are:

- **GET**: allow to know if the cookie attached to the request represents a valid session or not.
- **POST**: creates a new session. The *PIN* and the *timestamp* parameters are mandatory. The *timestamp* parameter is an integer in seconds.
- **DELETE**: forces to invalidate the session attached to the cookie.

B.0.4 `/api/v0/device/{id}/generate`

Used to generate a new password using the exposed functionality of the board. The supported HTTP methods are:

- **GET**: allows to obtain a new randomly generated password. The optional parameters are:
 - length**: the length of the password. The default value is 64.
 - upper**: boolean value that indicates if the password must contain uppercase letters. The default value is 1. Can be 0.

special: boolean value that indicates if the password must contain special characters. The default value is 1. Can be 0.

numbers: boolean value that indicates if the password must contain numeric characters. The default value is 1. Can be 0.

B.0.5 /api/v0/device/{id}/passwords

Used to manage passwords. The supported HTTP methods are:

- **GET:** allows to obtain the list of passwords. It supports the **hostname** parameter to filter the list of passwords by hostname. The **hostname** parameter is a string and it can be partial or complete. For example, if the **hostname** parameter is **mple.com**, then the list of passwords will contain passwords that have as hostname **www.example.com** or similar ones. Each password is represented by a JSON object with the following fields:

hostname: the hostname of the password.

password: the password.

username: the username.

id: the ID of the password.

- **POST:** allows to add and store in the board a new password. The parameters must be passed via the body in the form of a JSON object. The mandatory parameters are:

hostname: the hostname of the password.

password: the password.

username: the username.

B.0.6 /api/v0/device/{id}/password/{id}

Allows to manage a single password. The supported HTTP methods are:

- **GET:** allows to obtain the password record. The password is represented by a JSON object with the following fields:

hostname: the hostname of the password.

password: the password.

username: the username.

id: the ID of the password.

- **DELETE:** allows to delete the password.
- **PUT:** allows to update the password. The parameters must be passed via the body in the form of a JSON object, as the one to add a new password. The mandatory parameters are:

hostname: the hostname of the password.

password: the password.

username: the username.