Requirements: ”IoT Hardware Security Module Proof of Concept”

Autor: **Noli Manzoni**, **Sandro Tiago Carlao**

Project team: **Noli Manzoni**, **Sandro Tiago Carlao**

Version: **V0.1**

Date: **16.06.2017**

Bern University of Applied Sciences
Computer Science Department

Contents

1 Purpose of the document 1

2 Short description 1

3 Project objectives 1
3.1 BFH-Stakeholders . . . 1

4 System boundaries 2
4.1 Process level . . . 2
4.2 Functional level . . . 2
4.2.1 User stories . . . 3
4.2.2 Use cases . . . 4
4.3 Technical level . . . 10

5 Requirements 11
5.1 Sources and procedures . . . 11
5.1.1 Stakeholders . . . 11
5.1.2 Documents . . . 11
5.1.3 Systems in operation . . . 12
5.2 Functional requirements . . . 12
5.3 Technical requirements . . . 13
5.4 Quality requirements . . . 13

6 Glossary 14

7 Annex 14
7.1 Coordination of the requirements . . . 14
7.2 Definition of ready - checklist . . . 15

8 Version control 15

List of Figures

1 System Use Case Model . . . 2
2 HSM architecture using Raspberry Pi . . . 10
3 State of the art HSM software architecture . . . 10
4 RPiHSM software architecture . . . 11

List of Tables

1 Functional requirements . . . 12
2 Technical requirements . . . 13
3 Quality requirements . . . 13
4 Version control . . . 15

1 Purpose of the document

This document describes the objectives and the requirements of the project ”IoT Hardware Security Module Proof of Concept”. This document is written with LaTeX[1].

2 Short description

The goal of this project is to find out to which extent off-the-shelf Internet of Things (IoT) modules (like Arduino, Raspberry, etc.) can be programmed to function as Hardware Security Modules (HSM). We will build on such existing proofs of concept, improved and, if necessary, migrated to Java, ideally to obtain a functional HSM for personal use.

3 Project objectives

The main project objective is to pick, after a research and comparison, the IoT device with the larger and complete API and better compatibility with Java. After that we must test its limits as HSM. If it is possible, we should push these limits and, optionally, try to add external functionalities like remote access or a Trusted Platform Module (TPM). The secondary goal of this project is to give us the possibility to be more independent so that we can learn how prioritize our desires within the time frame that is at our disposal.

3.1 BFH-Stakeholders

- **Tutor:** Dr. Simon Kramer
- **BFH IoT promoter:** Prof. Peter Affolter
- **Security professor:** Prof. Gehrard Hassenstein

4 System boundaries

4.1 Process level

An HSM is a physical device that can come in various shapes and forms like smart cards, PCI cards, USB tokens and separate boxes that communicate over channels like TCP/IP, USB or RS-232, etc. These devices are designed to generate, store, protect, and manage digital keys. To use these devices a connection must be created, then with a specific software the crypto functionalities can be used.

4.2 Functional level

An IoT device must have a lot of functionalities to be an HSM (the most common are keys generation, file signing, data encryption and data decryption). The majority of the cryptographic functions can be implemented on a IoT device without problems thanks to the Keyczar library [7]. On other hand all the tamper proof functions and few others like TPM, RTC can be implemented only with external devices, for example data destroy in case of compromised system, or a tamper resistant key storage that could be achieved only with a specific hardware module.

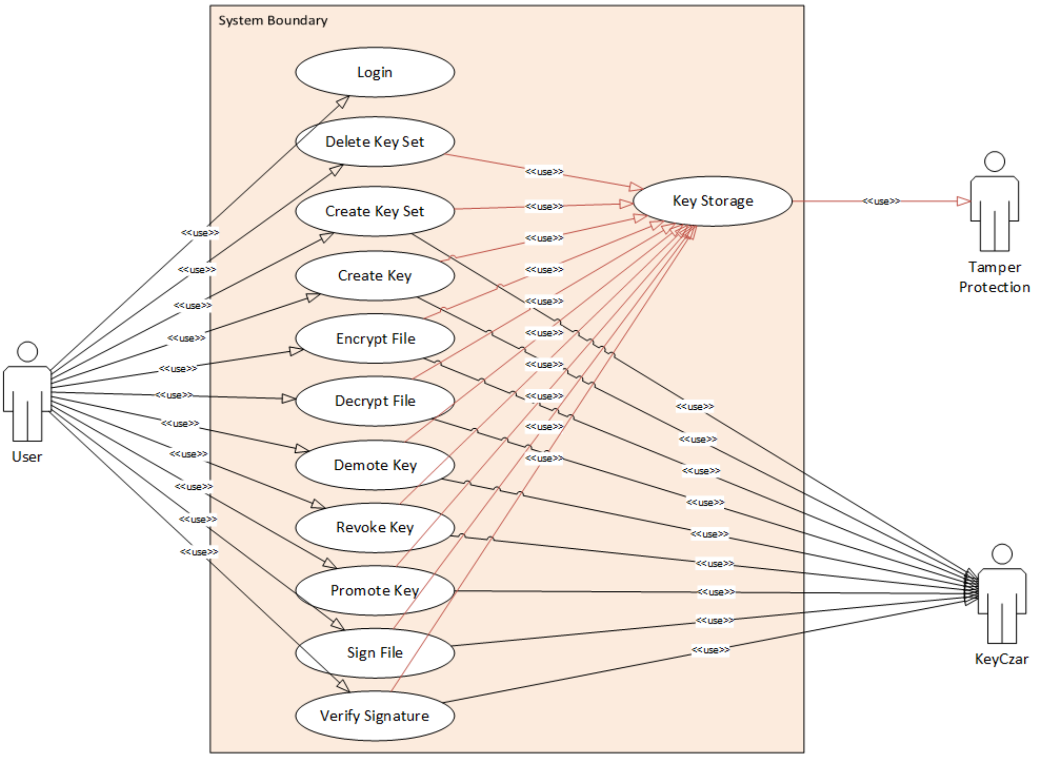

Figure 1: System Use Case Model

Actors:

- User: The user of the HSM
- Tamper Protection: External device or software implementation that increment security and integrity of the stored keys.
- Keyczar: Java library [7] that implement all the basic cryptographic operations.

4.2.1 User stories

Create key set

As a **User**, I want be able to create a key set, so that I can create a key.
Description: I enter the key set name, purpose and algorithm. The system check if the parameters are corrected.
Success: When I enter corrected parameters the key set is created.
Failure: When I enter incorrect parameters the key set is not created and an error message is displayed.

Create key

As a **User**, I want be able to create a key, so that I can perform different operation on it.
Description: I enter the key status and size. The system check if the parameters are corrected.
Success: When I enter corrected parameters the key is created.
Failure: When I enter incorrect parameters the key is not created and an error message is displayed.

Encrypt file

As a **User**, I want be able to encrypt a file with a given key set.
Description: I enter the file path and the name of the key set. The system check if the parameters are corrected.
Success: When I enter corrected parameters the file is encrypted.
Failure: When I enter incorrect parameters the file is not encrypted and an error message is displayed.

Decrypt File

As a **User**, I want be able to decrypt a file with a given key set.
Description: I enter the file path and the name of the key set. The system check if the parameters are corrected.
Success: When I enter corrected parameters the file is decrypted.
Failure: When I enter incorrect parameters the file is not decrypted and an error message is displayed.

Delete key set

As a **User**, I want be able to delete a key set.
Description: I enter the name of the key set. The system check if the parameters are corrected.
Success: When I enter corrected parameters the key set is deleted.
Failure: When I enter incorrect parameters the key set is not deleted and an error message is displayed.

Log-in

As a **User**, I want be able to log-in in the application, so that I access the system.
Description: I enter the user-name and the password.
Success: When I enter corrected credentials, I am signed in the system.
Failure: When I enter incorrect credentials, an error message is displayed.

Demote key

As a **User**, I want be able to demote a given key.
Description: I enter the key set name and the key version.
Success: When I enter corrected parameter, the key in the given key set is demoted.
Failure: When I enter incorrect parameter, the key is not demoted and an error message is displayed.

Revoke key

As a **User**, I want be able to revoke a given key.
Description: I enter the key set name and the key version.
Success: When I enter corrected parameter, the key in the given key set is revoked.
Failure: When I enter incorrect parameter, the key is not revoked and an error message is displayed.

Promote key

As a **User**, I want be able to promote a given key.
Description: I enter the key set name and the key version.
Success: When I enter corrected parameter, the key in the given key set is promoted.
Failure: When I enter incorrect parameter, the key is not promoted and an error message is displayed.

Sign file

As a **User**, I want be able to sign a given file with a given key set.
Description: I enter the file path and the name of the key set. The system check if the parameters are corrected.
Success: When I enter corrected parameters a signature is created.
Failure: When I enter incorrect parameters a signature is not created and an error message is displayed.

Verify a signature

As a **User**, I want be able to verify if a signature is valid with a given key set.
Description: I enter the file path, the signature path and the name of the key set. The system check if the parameters are corrected.
Success: When I enter corrected parameters the signature is checked and a success message is displayed.
Failure: When I enter incorrect parameters an error message is displayed.

4.2.2 Use cases

Create key set

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected.
Post-condition: A new key set is created.
Main success scenario:

1. The user enters the key set name, purpose and algorithm.

2. The system validates the purpose and the algorithm.
3. The system checks if the key set already exist.
4. The system creates a key set.

Extension:

2a The system determines that the purpose is not valid.

A The system displays an error message and the correct possibilities.

B The system waits for a new command.

2b The system determines that the algorithm is not valid.

A The system displays an error message and the correct possibilities.

B The system waits for a new command.

2c The system determines that the purpose do not correspond with the algorithm.

A The system displays an error message and the correct possibilities.

B The system waits for a new command.

3 The system determines that the key set already exists.

A The system displays an error message.

B The system waits for a new command.

Create key

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected.
Post-condition: A new key is created.
Main success scenario:

1. The user enters the key set name where the new key must be created, the key status and size.
2. The system validates the size and the status.
3. The system checks if the key set exist.
4. The system creates a key set.

Extension:

2a The system determines that the size is not valid.

A The system displays an error message and the correct possibilities.

B The system waits for a new command.

2b The system determines that the status is not valid.

A The system displays an error message and the correct possibilities.

B The system waits for a new command.

3 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Encrypt file

Scope: HSM cryptography phase.
Primary actor: User.
Precondition: The User is connected, the key set was created with crypt purpose and the given file exists.
Post-condition: A file is encrypted.
Main success scenario:

1. The user enters the key set name and the file path.
2. The system checks if the key set exist.
3. The system encrypts the given file.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Decrypt File

Scope: HSM cryptography phase.
Primary actor: User.
Precondition: The User is connected, the key set was created with crypt purpose and the given file exists and it is a encrypted file.
Post-condition: A file is decrypted.
Main success scenario:

1. The user enters the key set name and the file path.
2. The system checks if the key set exist.
3. The system decrypt the given file.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Delete key set

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected.
Post-condition: A key set is deleted.
Main success scenario:

1. The user enters the key set name.
2. The system checks if the key set exist.
3. The system deletes the given key set.

Extension:

2 The system determines that the key set not exists.

  A The system displays an error message.

  B The system waits for a new command.

Log-in

Scope: HSM authentication phase.
Primary actor: User.
Precondition: The User have an account.
Post-condition: The User is authenticated.
Main success scenario:

1. The user enters the user-name and password.
2. The system checks its credentials.

Extension:

2a The system determines that user account do not exists.

  A The system displays an error message.

  B The system wait for another authentication.

2b The system determines that password is not valid.

  A The system displays an error message.

  B The system wait for another authentication.

Demote key

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected and the given key is primary or active and it exists.
Post-condition: The given key is demoted.
Main success scenario:

1. The user enters the key path and the key version.
2. The system checks if the key set exist.
3. The system demotes the given key.

Extension:

2 The system determines that the key set not exists.

  A The system displays an error message.

  B The system waits for a new command.

Revoke key

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected and the given key is inactive and it exists.
Post-condition: The given key is revoked.
Main success scenario:

1. The user enters the key path and the key version.
2. The system checks if the key set exist.
3. The system revokes the given key.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Promote key

Scope: HSM management phase.
Primary actor: User.
Precondition: The User is connected and the given key is active o inactive and it exists.
Post-condition: The given key is promoted.
Main success scenario:

1. The user enters the key path and the key version.
2. The system checks if the key set exist.
3. The system promotes the given key.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Sign file

Scope: HSM cryptography phase.
Primary actor: User.
Precondition: The User is connected, the key set was created with sign purpose and the given file exists.
Post-condition: A signature is created.
Main success scenario:

1. The user enters the key set name and the file path.
2. The system checks if the key set exist.
3. The system create a signature for the given file.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

Verify a signature

Scope: HSM cryptography phase.
Primary actor: User.
Precondition: The User is connected, the key set was created with sign purpose and the given file exists.
Post-condition: A signature is checked.
Main success scenario:

1. The user enters the key set name and the file path and the signature path.
2. The system checks if the key set exist.
3. The system check is the signature is valid.

Extension:

2 The system determines that the key set not exists.

A The system displays an error message.

B The system waits for a new command.

3 The system determines that signature is not valid.

A The system displays an error message.

4.3 Technical level

After the study and the comparison (see attached document) we decided to use the Raspberry PI 3 model B.

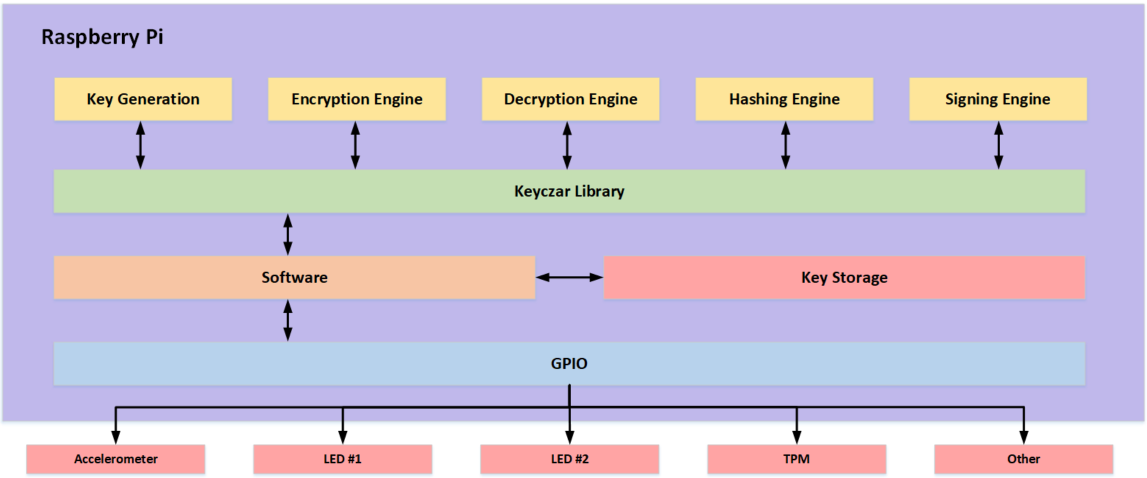

Figure 2: HSM architecture using Raspberry Pi

The current **state of the art** point out that this is the best system architecture available.

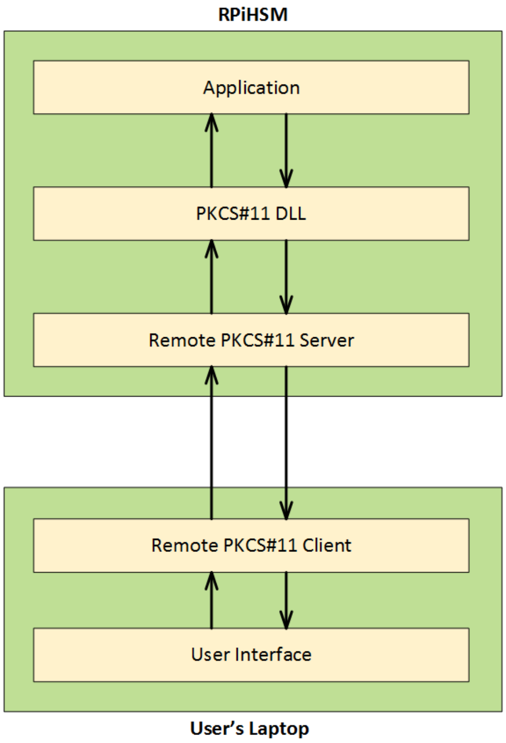

Figure 3: State of the art HSM software architecture

Due to the short time that is at our disposal this architecture is impossible to implement. For this reason we designed a lighter version that could be accomplished in a shorter time.

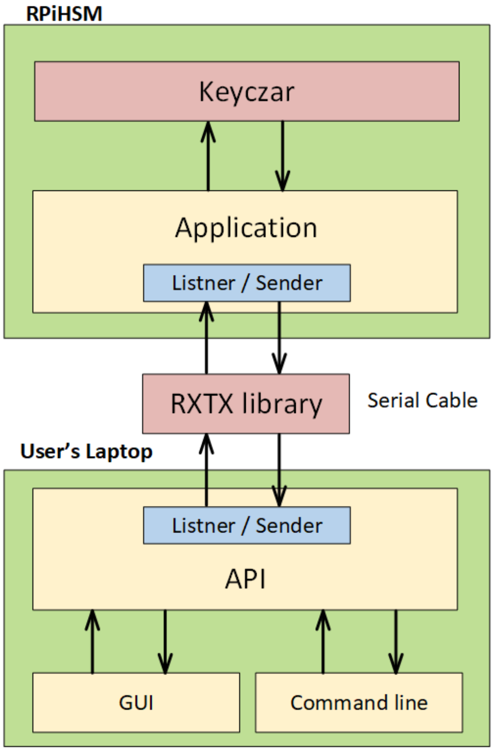

Figure 4: RPiHSM software architecture

5 Requirements

5.1 Sources and procedures

5.1.1 Stakeholders

In this project, as explained in the section 3, we have a lot of liberties because our tutor and the others stakeholders are here only to share their experiences and to give us some useful tips. Indeed we will take advantage of this to learn how to choose our objectives so that at the end we will have another ”Proof of Concept”.

5.1.2 Documents

Our project is based on three existing ”Proofs of Concept” [2][3][4] that we will use as examples. In addition to this, there are some standards as *FIPS PUB 140-1* and *FIPS PUB 140-2* that define the *Security Requirements for Cryptographic Modules* and the PKCS#11[5] that provides a standardized API for talking to cryptographic tokens.

5.1.3 Systems in operation

At this moment the world market is full of Hardware Security Modules that are certified and based on high performance hardware. However there are only few commercial IoT HSM product that are spread worldwide. The only one that we found is the Zymbit [6] that unfortunately can only be pre-ordered and therefore no reviews are available and therefore we cannot test its functionalities (**update 24.04.2017**: the Zymbit is now available and we are organizing to buy it. Now only few reviews are available and therefore we cannot make a conclusion).

5.2 Functional requirements

ID	**Status**	**Priority**	**Description**
F1			IoT
F1.1	Approved	Must	IoT module with the larger and complete APIs and better Java compatibility
F1.2	Approved	Optional	IoT module with lower programming level and higher control
F1.3	Approved	Optional	IoT module with higher tamper-protection possibilities
F2			HSM
F2.1	Approved	Must	HSM with the basic functionalities (private and symmetric key generation and storage)
F2.2	Approved	Must	HSM with best connection (as crypto card)
F2.3	Approved	Optional	HSM remote connection (web based)
F2.3	Approved	Optional	HSM with TPM functionality
F3			Architecture
F3.1	Approved	Must	Single-user-to-keys relationship
T3.2	Approved	Optional	PKCS#11 compatible software
T3.3	Approved	Optional	Multiple-user-to-keys relationships
F4			Connection
F3.1	Approved	Must	Best cable to create a connection between the PC and the device
T3.2	Approved	Must	Best programming language to transfer information between the PC and the device

Table 1: Functional requirements

5.3 Technical requirements

Some of these technical requirements are based on the decisions that we made on our preparatory analysis.

ID	Status	Priority	Description
T1			Device installation and configuration
T1.1	Approved	Must	Use KeyCzar as cryptographic library
T1.2	Approved	Optional	Implements standardized PKCS#11 interfaces
T2			Implement connection
T2.1	Approved	Must	Implements a connection between the user application and the Raspberry Pi using a USB serial cable
T3			Use Keyczar cryptographic functionalities
T3.1	Approved	Must	Generates key sets for various purposes (encryption, decryption, signing)
T3.21	Approved	Must	Generates keys with various size
T3.3	Approved	Must	Encrypts files with different algorithms (AES, RSA)
T3.4	Approved	Must	Decrypts files with different algorithms (AES, RSA)
T3.5	Approved	Must	Signs files with different algorithms (HMAC, DSA, RSA)
T3.6	Approved	Must	Verifies signatures
T3.7	Approved	Must	Deletes key sets
T3.8	Approved	Must	Promotes, demotes and revokes keys
T4			Graphic User Interface (GUI) on client
T4.1	Approved	Must	Allows the client application to communicate over serial connection with the Raspberry Pi
T4.1	Approved	Must	Uses the application via command line
T4.1	Approved	Optional	Uses the application via GUI

Table 2: Technical requirements

5.4 Quality requirements

ID	Status	Priority	Description
Q1			Assure CIA triad
Q1.1	Approved	Optional	Assures the confidentiality focusing on a crypted transmission channel
Q1.2	Approved	Optional	Assures the integrity of the sent directives by using private/public keys communication
Q1.3	Approved	Optional	Assures that if a disruption of communication appear the user is informed and the data is saved correctly
Q1.4	Approved	Optional	Tries to create a secure key storage with a Public Key Infrastructure (PKI)
Q2			Performance
Q2.1	Approved	Optional	Assures fast communication between user and HSM
Q2.1	Approved	Optional	Tries to optimize the boot of the Raspberry OS and HSM software

Table 3: Quality requirements

6 Glossary

IoT:
The Internet of things (IoT) is the inter-networking of physical devices embedded with electronics, software, sensors, and network connectivity that enable these objects to collect and exchange data.

HSM:
A hardware security module (HSM) is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing.

USB Tokens:
An USB token is a physical device that is used to establish personal identity without use of a password to access a network.

TPM:
Trusted Platform Module (TPM) is an international standard for a secure cryptoprocessor.

Rs-232:
In telecommunications, RS-232 is a standard for serial communication transmission of data.

RTC:
A real-time clock (RTC) is a computer clock that keeps track of the current time.

PKCS#11:
The PKCS are a line of standards created by RSA laboratories that provide specifications concerning cryptography. For our HSM we are interested in the number 11, also called Cryptoki, which provides standardized API for talking to cryptographics tokens.

FIPS PUB:
Federal Information Processing Standards (FIPS) are publicly announced standards developed by the United States federal government for use in computer systems by non-military government agencies and government contractors.

7 Annex

No annexes.

7.1 Coordination of the requirements

No conflicts.

7.2 Definition of ready - checklist

- □ The value of the task is clearly articulated.
- □ Task details are sufficiently understood by the team.
- □ Task dependencies are identified
- □ The task is sufficiently small to be completed in the given time.
- □ Acceptance criteria are clear and testable.
- □ Performance criteria, if any, are defined and testable

8 Version control

Version	Date	Description	Autor
X0.1	21.02.2017	Document creation	Noli Manzoni and Sandro Carlao
X0.2	27.02.2017	Vision and objectives	Noli Manzoni and Sandro Carlao
X0.2	06.03.2017	System boundaries	Noli Manzoni and Sandro Carlao
X0.3	15.03.2017	Functional Requirements	Noli Manzoni and Sandro Carlao
X0.4	21.03.2017	Technical & Quality Requirements	Noli Manzoni and Sandro Carlao
X0.5	06.04.2017	Glossary and Definition of Ready	Noli Manzoni and Sandro Carlao
X0.6	23.04.2017	Information improvement	Noli Manzoni and Sandro Carlao
X0.7	24.04.2017	Zymbit update	Noli Manzoni and Sandro Carlao
V0.1	22.05.2017	Document final review	Noli Manzoni and Sandro Carlao

Table 4: Version control

References

[1] *LaTeXis a typesetting system designed for the production of technical and scientific documentation*
`https://www.latex-project.org/`

[2] *Library to use an Arduino Due as Hardware Security Monitor for Amazon Web Services*
`https://github.com/st3fan/arduino-aws-hsm`

[3] *Article about the use of Arduino as Hardware Security Module*
`https://randomoracle.wordpress.com/2013/01/15/arduino-tpms-and-smart-cards-redefining-hardware-security-module`

[4] *Guide to build a Raspberry Pi HSM for RSA 2014*
`https://cryptosense.com/building-a-raspberry-pi-hsm-for-rsa-2014/`

[5] *PKCS#11 cryptographic token interface standard*
`https://en.wikipedia.org/wiki/PKCS_11`

[6] *Security Module for Raspberry Pi*
`https://www.zymbit.com/zymkey/`

[7] *Keyczar Java Library*
`https://github.com/google/keyczar`