BERN UNIVERSITY OF APPLIED SCIENCES

Project 1

Module: BTI 7301

IoT Hardware Security Module Proof of Concept

Students

Noli Manzoni, Sandro Tiago Carlao

Tutor

Dr. **Simon Kramer**

Declaration on the intellectual property

I the undersigned **Noli Manzoni** declare that:

- I know the ”guidelines on the management of plagiarism at the Bern University of Applied Sciences” and the ”HAFL code of ethics on the use of information sources”, as well as the consequences of their non-compliance.
- I have complied with it in the realization of this work.
- I have realized this work personally and independently.
- I accept that my work will be tested using a software of detection of plagiarism and kept in the database of the HESB.

Signature

____

Place, Date

____

I the undersigned **Sandro Tiago Carlao** declare that:

- I know the ”guidelines on the management of plagiarism at the Bern University of Applied Sciences” and the ”HAFL code of ethics on the use of information sources”, as well as the consequences of their non-compliance.
- I have complied with it in the realization of this work.
- I have realized this work personally and independently.
- I accept that my work will be tested using a software of detection of plagiarism and kept in the database of the HESB.

Signature

____

Place, Date

____

Abstract

The risk of storing cryptographic keys in an hard disk or even in memory is constantly increasing because once the system they are stored in is compromised, the keys must be replaced. Therefore, to find a solution to this problem new security module such as smart cards or HSM were created. Unfortunately these devices were designed only for commercial use and so, private users were abandoned without any possible solution. For this reason the goal of this project is to find out which extent off-the-shelf Internet of Things modules can be programmed to function as a Hardware Security Modules.

Contents

List of Figures

1 Purpose of the document

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

2 Description

2.1 Project Goal

The goal of this project is to find out which extent off-the-shelf Internet of Things (IoT) module (e.g. Arduino, Raspberry) can be programmed to function as a Hardware Security Module (HSM).

An HSM is a cryptographic device for secure:

- Private and symmetric key generation and storage (protection from key compromise)
- Algorithm execution (protection from side-channel attacks).

You will build on such existing proofs of concept, improving and, if necessary, migrating them to Java, ideally obtaining your own functional HSM for your own personal use as a result of your Project-1 work.
Pick the most suitable IoT device and test its limits as HSM. If possible, push these limits and, optionally, try to add external functionality like remote access or Trusted Platform Module (TPM) [5].

2.1.1 Existing proofs of concept

Arduino:

- Arduino HSM for Amazon Web Services [2]
- Arduino, TPMs and smart cards: redefining Hardware Security Module [3]

Raspberry Pi:

- Building a Raspberry Pi HSM [4]

2.2 BFH-Stakeholders

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

3 Assignment

After the first meeting with our tutor, we have a better idea about the project.
First of all, we must search for other possibly existing “Proofs of Concept” in addition to the three that we have received. In order to be sure about the **state of the art** we must analyse this information. Lastly we must study the functionality of these IoT devices and then decide which IoT device is the most suitable for this project.

IoT module with higher priority:

- IoT module with good APIs
- IoT module with higher compatibility with Java to use Keyczar [6]
- **Optional**: IoT module with lower programming level and higher control
- **Optional**: IoT module with higher tamper-protection possibilities

During this decision phase we must contact the other stakeholders to have more information.
Initially, we will contact the BFH IoT promoter *Prof. Peter Affolter [7]* to have a better understanding about the IoT technologies and their functionalities. After this meeting we should decide which IoT module we are going to use.
Lastly, we will contact *Prof. Gehrard Hassenstein [8]* to have information about a Bachelor project that is related with ”IoT Security” and to have feedback about his experience in this domain.

3.1 Meetings

Date	**Stakeholder**	**Aim**
28.02.2017	Dr. Simon Kramer	Project introduction
07.03.2017	Prof. Dr. Peter Affolter	IoT technologies
13.03.2017	Kipfer Heinz	Raspberry Pi 3 acquisition
16.03.2017	Prof. Dr. Gehrard Hassenstein	Security introduction
22.03.2017	Dr. Simon Kramer	Requirements revision
27.03.2017	Kipfer Heinz	FTDI Cable acquisition

4 Hardware Security Module

In addition to what we explained in the section 2.1 an HSM module can be internal or external and it can be connected to a computer or a server.
The principal use of the HSM is to outsource the cryptographic functions of a computer system to ensure that these critical operations are made in a safe environment.

4.1 Architecture

As previously said, the HSM are designed to protect cryptographic keys. Therefore, they have a multi-level architecture as shown in the Figure 1.

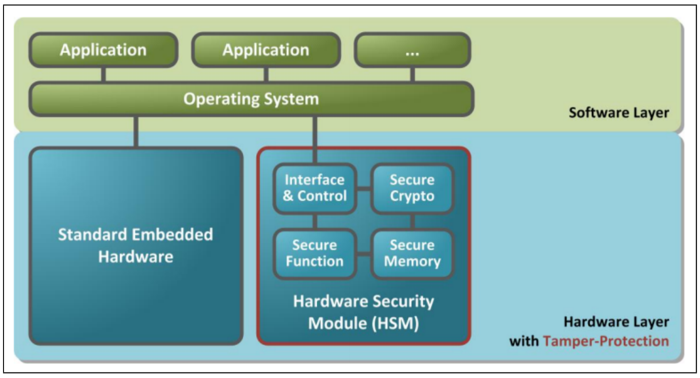


Figure 1: HSM architecture (image source: see [9])

Secure memory is a non-volatile data storage that stores critical information like cryptographic keys and certificates. This memory is protected against side channel attacks to prevent unauthorized use of these confidential data.

Secure cryptography is a software section that manages all the cryptographic algorithms used for encryption and decryption (AES[10], 3DES[11], Camellia[12], RSA[13] etc.), key generation, key verification and all the others cryptographic activities.

In the section **Secure function** are stored the other functions that are not related to cryptography, for example system protection functions such as a physically protected clock signal or an internal random number generator.

In the area **Interface and control** there is the HSM logic, like the system APIs that are used from the external world to access the HSM functions.

The **Tamper-protection** layer has the task of protecting all the logical system from external attacks such as non-authorized data manipulation. This protection is implemented with, in some case, tamper-protection algorithm and always with special shielding or coatings.

5 Proof of Concept

To create our HSM we must discover the limits of these IoT modules and to reach this objective, we must search for others ”Proofs of Concept”, related articles and similar commercial products.

5.1 Arduino

The main reason that lead *Stefan Arentz* develop this Aruduino Due HSM [14] was that storing secret keys, such as Amazon Web Services (AWS) key, in a configuration file generates a big security problem. The base idea of the author of this article was to delegate to the IoT device the signing of Amazon Web Services API requests because it was quite powerful. For him this was a good idea because the only possibility to read the AWS key was to stole the device and use an electron microscope to read the CPU values.

5.2 Raspberry

In this article [4], the Cryptosense team [15] explains how they have built a HSM with a Raspberry Pi for their demonstration at the RSA Expo 2014 [16]. In this conference, the team wanted to show how a costumer can use their Cryptosense Analyzer to audit, configure and secure a HSM. To simulate all existing HSM on the market the team decided to build its own module. The idea was to have a PKCS#11 [17] daemon on the Raspberry Pi, so that it could be configured and used to find a safe configuration for which Cryptosense Analyzer could not find any attacks. To simulate the PKCS#11 they use an open-source library named Opencryptoki [18] that they have expressly changed so that they could not use a function named C_CreateObject, which allows an attacker to import his own keys. After the RSA Expo the Cryptosense team realaised that their HSM was not the most secure because it would not resist any physical attacks but it was a much more portable device than others HSM on the market.

5.3 Other

5.3.1 CryptoCape the Beaglebone security daughterboard

CryptoCape is the Beaglebone’s first dedicated security daughterboard and it was made in collaboration with the hacker Josh Datko. For what we have found, this module with the Beaglebone Black motherboard can be used like an Arduino or a Raspberry Pi to create a HSM. However this additional module can add more security because it is designed to perform cryptographic operations. The major functionality is the Trusted Platform Module [19] that can be used to store cryptographic keys.

6 Meetings

6.1 *Prof. Peter Affolter*

In this meeting, we talked about the three most suitable IoT modules for this project (Arduino, Raspberry Pi and Beaglebone).

After some research and the discussion, we found out that the first option (Arduino) has a great environment but it lacks ins specific functionalities. In other word, and to cite *Prof. Peter Affolter*, Arduino is an "artist" module that is useful for students but not for computer science engineers. However this module has a low energy consumption and thus it has a great mobility. Moreover it has an easy programming language that allows fast learning and fast implementation. Nonetheless, an HSM cannot be developed in a limited system like this.

The second module (Raspberry Pi) could be a good choice in our opinion, considering that it has great hardware (for its price) and a less limited operation system. The only problem of this IoT device according to *Prof. Peter Affolter* is that it has a high-energy consumption (2.5 A, 5V). Consequently, this could decrease the HSM mobility.

We have not found sufficient information about the Beaglebone IoT module, but with its CryptoCape, could be a good piece of hardware to test.

We conclude that Raspberry Pi could be an excellent IoT device to create a HSM but, actually, there is a better option. The alternative is a combination of the Raspberry Pi and the Beaglebone with his CryptoCape module. According to *Prof. Peter Affolter*, this is the best option, because the Raspberry Pi offers a lot of possibilities and is well documented. On the other hand, it has not particular protection system. That is where the Beaglebone comes in with its security module that is a great place to experiment.

6.2 *Prof. Gerhard Hassenstein*

In this meeting we realized that there is a little bit of confusion about what we should achieve in this project. To overcome this problem, according to *Prof. Gerhard Hassenstein*, we must focus only on one of these options.

The first one is that this IoT device will become a real HSM. Therefore, this means that it should manage multiples keys using a user to keys relationship. This involves that the HSM must be accessible by multiple person at the same time with, for example, a web interface.

The second option is to use this device as a crypto card or, with other words, an USB HSM. This means that there is only a single user to keys relationship. Furthermore, if this option is chosen we should decide if this IoT device must be PKCS#11 compatible or not.

According to the project description and early *Dr. Simon Kramer* comments, the project emphasis is on ”what we want to do” within the time frame that is at our disposal, therefore our first priority is the single user HSM. To achieve this objective the first step to do is to decide how we will build this system. The two possibilities are the following:

- Server/HSM side application PKCS#11 compatible and a client side application that use a PKCS#11 Java wrapper to communicate with the hardware (overall compatible).
- Server/HSM side application that communicate with a client side application without any PKCS#11 standard.

6.2.1 PKCS

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.

7 IoT devices security

For this project the IoT device security is the most important thing because we are trying to move all the cryptographic functions of a PC to an external device. Therefore we must be careful in our design. The principal thing that we must think about, is that we can prove our application is insecure, but we can't prove our application is secure. This lead us to understand that we must try to keep our device off the network and other possible source of dangers as much as possible.

To find out how many layers of constraints the device must have, we must assign a value to the information that are stored in it. To do that we must understand how much the data are relevant for the final user.

To achieve a good outcome, we must build our application on the three basic security components, that are: confidentiality, integrity and availability (the CIA triad[29]).

Confidentiality is defined as concealment of information but we must consider that, if the device can be physically accessed, all information can be read with a microscope. For this reason, the focus is on protecting the information transmission channel with the cryptography.
Integrity is making sure that the directives sent to the IoT device has not been changed along the road. We do not care if a hacker looks at the data bust we make sure that a hacker cannot forge the data. All of this can be achieved with a cryptographic hash on the message and, if we want to be sure that the data has not been forged using a man-in-the-middle attack, we must use a public/private keys communication.
Availability means to be able to handle a disruption of communication service by communicating it to the user. In our case this disruption can be caused by electrical noise.

To reach the above objectives we must take into account key management. There are two approaches to this topic and thhe first is hardware key management. Companies like Infineon[30] build chips that include private and public keys but these externals components are expensive and using one of them will double the price of a IoT device. The second method is to use software to distribute keys during the configuration. These methods are called PKI (Pubic Key Infrastructure)[31].

8 Comparison

In this section we will present our research, comparison and choice of the different technologies that we had available for this project.

8.1 Hardware

8.1.1 Arduino

Arduino is a micro-controller, developed in Italy in 2005 by *Massimo Banzi, David Cuartielles, Tom Ingoe, Gianluca Martino and David Mellis* at the *Interaction Design Institute* in Ivrea. As part of the Arduino project, there is also an open-source development environment to interact with the device and its extensions (Modules, Shields, Kits or Accessories).

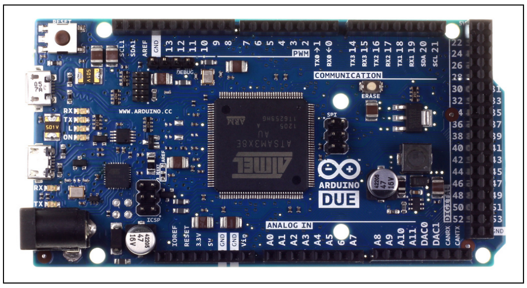


Figure 2: Arduino Due

Java compatibility The Arduino come with LininoOS pre-installed [32] and the main programming languages are C, C++ and Python. However, Java ME and Java Embedded can both run on it without problems. For this reason Arduino has some GPIO libraries develped in Java (e.g. *RXTX Java Library* [33] and *JArduino Library* [34]).

Tamper protection As tamper protection *SparkFun* offers an Arduino crypto shield [35] that adds specialized hardware and software that performs various operations such as real time clock, trusted platform module, etc.

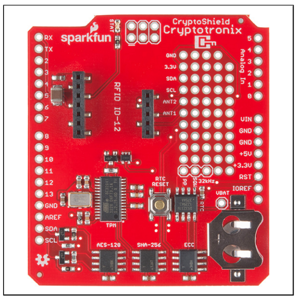


Figure 3: Arduino crypto shield by Sparkfun

8.1.2 Raspberry Pi

Raspberry Pi is a single-board computer developed in the UK by the Raspberry Pi Foundation. The first Raspberry Pi has been sold on 29 February 2012 and now there are over 10 million modules over the world.

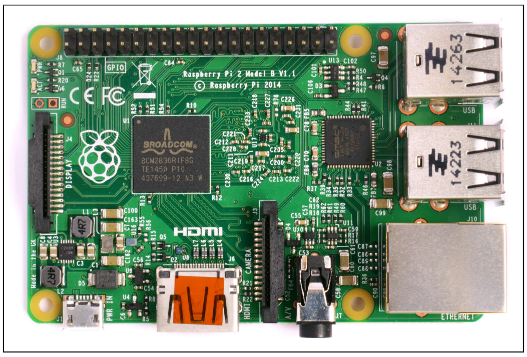


Figure 4: Raspberry Pi Model B+

Java compatibility Raspberry Pi V3 has a pre-installed operating system called Raspbian [22] that comes with plenty of software including Java ME. However, Raspbian does not have his own GPIO [20] library. The only library that provides a friendly object-oriented Java I/O API is Pi4J[21].

Tamper protection Tamper protection for this cheap IoT device is not so important because the normal users do not care about the physical security. Therefore, there are only few possibilities. The first one, and the easiest one, is to buy a pre-build security module with all the functionalities such as Zymbit 3i I2C that unfortunately can be only pre-ordered. (**update:** the Zymbit is now available (24.04.2017) to more info see section 10) [23]. The second option, and the more difficult one, is to try to add an external module such as CryptoShiled. According to this article [24] any board with the Arduino form-factor can be attached to this module consequently with some prototyping the Raspberry Pi could use it.

8.1.3 BeagleBone Board

Beaglebone is an open-source single-board computer developed by Texas Instruments. The last model named Beaglebone Black is compatible with Ubuntu and Android.

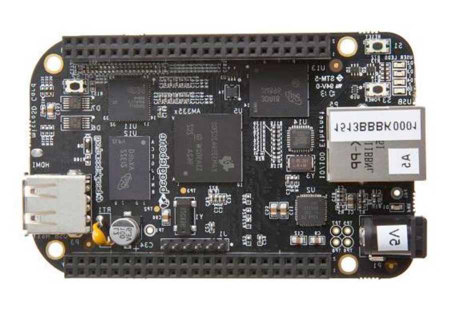


Figure 5: Beaglebone Black

Java compatibility Java in this IoT device is not a problem because the Beaglebone can run Ubunto or the Android OS without problems. Unfortunately, Bradcom has not released a compatible Android image yet. Hence, we discard the Android possibility. In any case, we have, like Raspberry Pi, the possibility to install Java ME [25] [26]. The Beaglebone official GPIO library is developed in Javascript therefore is not suitable for us. Our choice is Bulldog[27] because it is the best GPIO Java Library of the few available on the internet.

Tamper protection The big advantage of this module is a dedicated daughterboard called CryptoCape [28]. This cape adds specialized hardware and software that performs various operations such as real time clock, trusted platform module, etc.

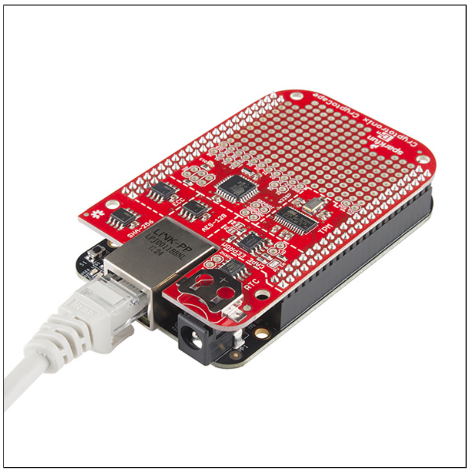


Figure 6: Beaglebone Black with CryptoCape (image source: see [28])

8.1.4 Choice

After our meeting with *Prof. Peter Affolter* (see section6.1) we decided that our primary objective, for now, is to create a basic HSM with a Raspberry Pi module. Secondly we must check if is possible to connect it with the Beaglebone. If this will become something useful, with *Prof. Peter Affolter*'s help, we could go a step forward by prototyping an adapter for these two modules.

8.2 Connection

There are many ways to connect Raspberry Pi to a client, each type of connection has its own advantages and disadvantages. Here are presented the most used connections available in the market.

8.2.1 FTDI Cable

The most used serial connection to control a Raspberry Pi, is the FTDI cable (or console cable). Many projects have been done using this cable and therefore there is a lot of documentation available that explains how to use it [41].

Figure 7: FTDI Cable (image source: see [47])

8.2.2 Ethernet Cable

Another good choice could be the Ethernet cable because no external adapters or special cables are required and no additional power voltage is needed. In addition to that the speed can reach 100 Mbps.

Figure 8: Ethernet Cable (image source: see [48])

Here[49] is a good example of Client-Server application that is implemented with an Ethernet cable. Unfortunately there is a problem, the above cited article use a socket written in C therefore the Java compatibility is not confirmed.

8.2.3 RS-232

RS-232 is the predecessor of USB, and it has been replaced because of its big interface, low speed and high voltage consumption (5V).

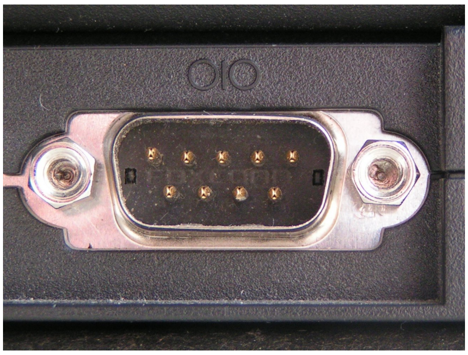


Figure 9: RS232 Interface

The procedure to connect the Raspberry Pi is similar to the FTDI Cable. For the RS-232 install process this guide [51] is a good reference.

8.2.4 Choice

For our project we decided to use the USB FTDI Cable because the others options are not suitable for our HSM use. As already said the RS-232 has been replaced so this is rejected in first place, second the Ethernet may have been a good choice but in the last few years a lot of laptop are sold without any Ethernet port so that they can be thinner.

9 USB HSM

To create this device, we will build upon the Raspberry Pi Proof of concept [4]. Therefore the architecture of our implementation will be very similar to the one on the article.

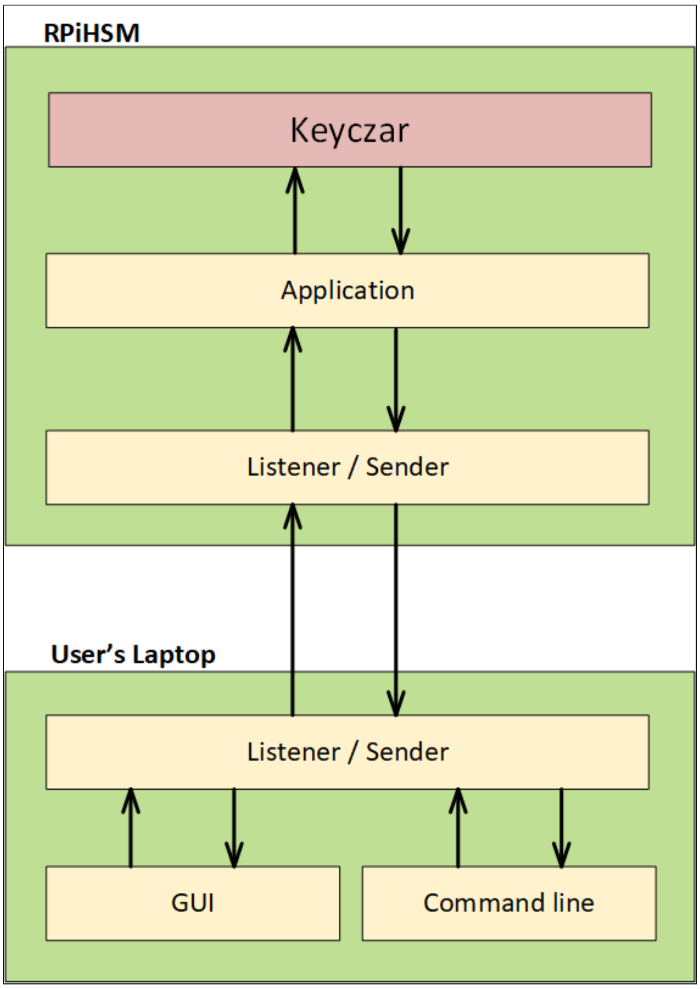


Figure 10: Architecture sketch based on [4]

As shown in the figure 10 an ideal application should have two distinct parts. The client application (user's laptop) should have a simple UI that will allow the user to use all RPiHSM functionalities. This is possible because the program should also have a remote PKCS#11 client that will establish secure connection between the user and the Raspberry Pi. Therefore in the second part of the ideal application (RPiHSM) there will be a PKCS#11 server that will manage all the incoming requests by checking that the received directives are not been changed along the road (see section 7). Once the message integrity is checked the PKCS#11 server will forward it to the final application by using a DLL file. When the request will reach the end stage it will be process and then a replay message will send back to the user using the same path. The client application should be based on specific PKCS#11 Java Wrapper like [52] so that it will be compatible with all sort of PKCS#11 application.

10 Zymbit

Zymbit[23] is a security module for the Raspberry Pi that became available in march 2017. This device is a enhanced version of what our final product should be because it has more functionalities e.g. creation of an unique ID token using host device specific measurements, detection of anomalies like brown-out[54] events and orientation change, use of battery-backed real time clock, etc. Therefore this product is not suitable for this project because it have all required functionalities with simple API that could be used to create an application in a very short time. This product anyway could be used in a future project that aim to have a web-based RPiHSM or a Raspberry Pi application what need high security.

11 Design

11.1 User stories

11.1.1 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 incorrected parameters the key set is not created and an error message is displayed.

11.1.2 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 incorrected parameters the key is not created and an error message is displayed.

11.1.3 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 incorrected parameters the file is not encrypted and an error message is displayed.

11.1.4 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 incorrected parameters the file is not decrypted and an error message is displayed.

11.1.5 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 incorrected parameters the key set is not deleted and an error message is displayed.

11.1.6 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 incorrected credentials, an error message is displayed.

11.1.7 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 incorrected parameter, the key is not demoted and an error message is displayed.

11.1.8 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 incorrected parameter, the key is not revoked and an error message is displayed.

11.1.9 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 incorrected parameter, the key is not promoted and an error message is displayed.

11.1.10 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 incorrected parameters a signature is not created and an error message is displayed.

11.1.11 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 incorrected parameters an error message is displayed.

11.2 Use cases

11.2.1 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.

11.2.2 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.

11.2.3 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.

11.2.4 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.

11.2.5 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.

11.2.6 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.

11.2.7 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.

11.2.8 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.

11.2.9 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.

11.2.10 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.

11.2.11 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.

For our RPiHSM we decided to adapt the system architecture shown in the figure 10 so that it is suitable for our project. As illustrated in the figure 11 we will divide our program in three main parts.

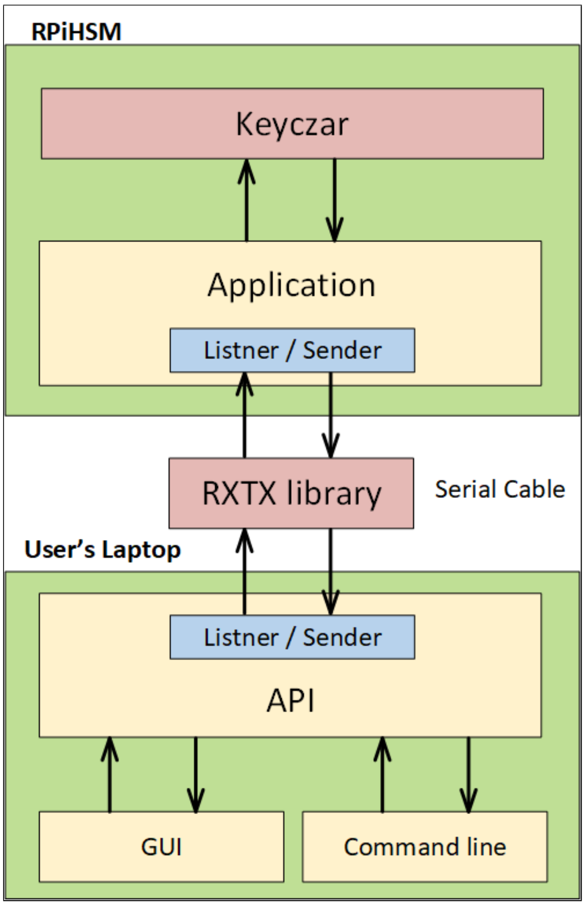


Figure 11: Applications design

11.3 Raspberry Pi Application

SPIEGAZIONE PERCHE e come

11.3.1 Log-in

To have higher security on this devices we decided to authenticate the user using the Linux Pluggable Authentication Modules (PAM) [57]. With this solution, as well as the security improvement, we can manage multiple users. To implement this standard security architecture in Java we decided to use a Java-PAM bridge called JPAM[58].

11.4 API

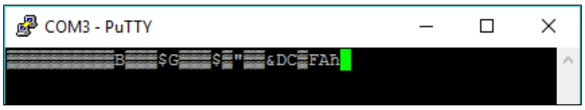
11.4.1 Serial Communication

Now, after that we chose all the hardware component, we must analyse the different programming languages to find out which one is the most appropriate. For this project we had two major idea, the first one was to create a user application using a native system, like *PowerShell* for Windows and *Bash* for Mac and Linux, so that the HSM could be portable and less dependent on external software. The second idea is to use Java and his rxtx library [33] so that the application could be compatible for all platform.

Native System - PowerShell The main reason that lead us to choose this language was to allow the user to use our HSM without any external software. Moreover, PowerShell is one of the most high-level language that allow the programmer to do very low operations with the hardware. To create a connection this language makes available a special object that allows the user to write and read from the console.

```
$port= new-Object System.IO.Ports.SerialPort COM3,115200,None,8,one
$port.Open()
#linux command to show network cards configuration
$port.WriteLine("ifconfig")
#read if someting has been writing
$port.ReadExisting()
$port.Close()
```

When we created the connection unfortunately, we had some problems. When the connection was established with the Raspberry Pi using putty, the user must click various times the button enter so that the device realizes that someone is trying to talk to it.



```
▒▒▒▒▒▒▒▒▒▒B▒▒▒$G▒▒▒$▒"▒▒&DC▒FAh
```

Figure 12: Strange characters on Putty

This problem disappeared when we solved the problem of the speed (see section 12.1.2) so, we decided to improve our script to see how an user could log-in.
For security reason, we decided that if the PowerShell script is launched when an user is already logged in the Raspberry Pi, the session is interrupted and he needs to re-authenticate (see integral code (line 1-36): *RPiHSM/General/Native-PowerShell/RPiHSM-Authentication.ps1*).

This application besides creates a connection, must also be able to send and receive files from the Raspberry. Initially, we tried to write each single byte of the file on the Raspberry Pi console but we realized that was really slow and that it worked only for text files. The files were slowly transferred because each character must be converted in hexadecimal, transferred throw the Raspberry Pi I/O layer and then physically printed on the console. Besides, images could not be transferred because the values were translated to ASCII code giving an incorrect image (see integral code (line 39-56): *RPiHSM/General/Native-PowerShell/RPiHSM-Authentication.ps1*).

To solve this problem and have better performance we decided to disable the serial console on the Raspberry Pi and create a Python listener in the Raspberry Pi that waited for data in the console. This solution unfortunately gave us another problem indeed, we could send files and images to Raspberry but we could not log-in anymore. To solve this setback we found a Python module [56] that allow us to ask the OS if the password for a given user was correct (see integral code: *RPiHSM/General/Native-PowerShell/RPiHSM-Python_Listener.py* and *RPiHSM/General/Native-PowerShell/RPiHSM-Authentication_and_File_Transfer.ps1*).

Java Application To create a serial connection with Java we found two product. The first one, that we will use in the "client-side", application is the already cited Java wrapper library RXTX. For the RPi application we found a good documented Java library called Pi4J [21] that provides a friendly object-oriented I/O API to access the full I/O capabilities of the device. To use the last library we just added the dependency on Maven and we had not problem because it had already inside the RXTX libraray. To use this one indeed, we had some problems. To make it work we added two DLL files (rxrxSerail.dll and rxtxParallel.dll) and a jar in the Java home directory. The final code *RPiHSM/General/Java/* is a object oriented version of the found examples on the Pi4J and RXTX website.

Conclusion To implement our application we decided to use Java because it allow us to make more controls on the client side (the program speed up because if there are errors they are not generated after that the information are passed thought the serial cable) and because the Java library are more stable then the others one described in the power-shell paragraph.

11.5 Command line

11.5.1 Command handling

The "client-side" application must be designed so that the user can use all Keyczar functionalities. Because the Keyczar has a lot of possible commands and maybe it will be updated in the future we decided to design our application so that it can be flexible. To do this we use the Factory Pattern to return the right Keyczar object that implement an interface that have a common execute method so that the application can be expanded without problem. To prevent that the Keyczar library generate error that could not be caught (they are generated in an external library) we decide to use a commend line arguments parser framework called JCommand. This library give us, stable and tested code, that allow us to check if all needed information are received.

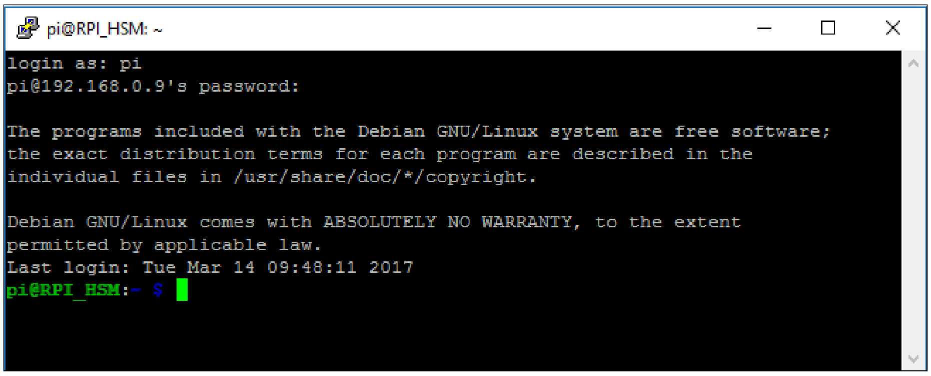
12 RPiHSM

12.1 Raspberry Pi 3 Configuration

In this section we will explain how to configure the Raspberry Pi 3 model B so that it can work with Java and why we choose the relative software.

12.1.1 OS installation and configuration

First of all, for this project we choose the Raspbian OS because due to the fact that it is the foundation's official operating system it is one of the most secure OS available. The Raspbian OS must be downloaded from the Raspberry Pi website and then it must be installed on the SD card by following the relative instruction. Once the OS is installed the SD card must be inserted in the apposite slot and then the Raspberry Pi must be turned on. To connect to it we decided to use the SSH protocol [36] because it ensure a fast remote access. To activate this protocol the right interfaces must be turned on by following this guide [37].



```
login as: pi
pi@192.168.0.9's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 14 09:48:11 2017
pi@RPI_HSM:~ $
```

Figure 13: SSH connection via PuTTY

To transfer data from our pc to Raspberry Pi we chose the command scp[38] because it is built purely for file transfer and in terms of speed is generally much faster then his competitors.

```
C:\project>scp RPI_test.jar pi@192.168.0.9:project
```

12.1.2 FTDI Cable configuration

To configure the FTDI the first thing to do was to find which driver our computer needed to establish a serial connection. To discover the right driver, we searched the brand and model of our cable then, we searched on the manufacture website the newest driver to install [40]. The second step, for the found documentation [41] was to modify the */boot/config.txt* file on Raspberry Pi so that the system unlocks the serial port.
Unfortunately, after that we connected the FTDI cable with the raspberry by following this schema 14 we realized that something was not going as predict. Indeed, we could connect

to the Raspberry Pi console using Putty [42] only with the speed of 9600 bps, and this was not a suitable option because a file of 44 KB could be transfer within 8 - 10 minutes. To resolve this problem, we searched a lot in the web and finally we found a solution [43]. The key of this dilemma was that the Raspbian was not configure to work with the maximal speed 115200bps, therefore we modified the file cmdline.txt by adding the right number.

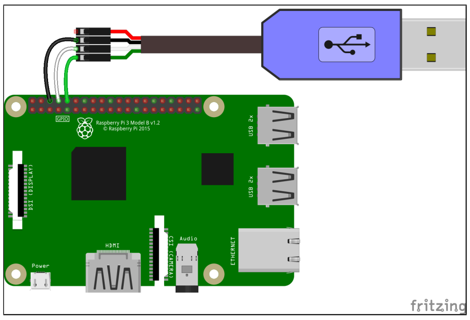


Figure 14: FTDI cable to RPi GPIO connection schema

12.2 Google Keyczar configuration

In the configuration of Google Keyczar we had some problem because it is not well-known and therefore few documentation are available. The first step to use this library is to clone it from the Github project [6] and this far no problems. Logically the second step should be the compile phase but when we try a lot of compile errors appear. After various attempts we found that a new branch with the last library version as name must be created.

```
git checkout -b Java_release_0.71j
```

After this the compilation of the source code so that it can be installed on Maven[39] was not anymore a problem. Indeed only the following command must be executed in the folder *java/code*.

```
mvn -e clean compile test package
```

To complete the process, the generated jar must be installed on Maven and the right dependency must be added in the *pox.xml* file.

```
Mvn install:install-file -Dfile=keyczar-0.71j-031417.jar
-DgroupId=org.keyczar -DartifactId=keyczar
-Dversion=0. 71j  -Dpackaging=jar

<dependency>
        <groupId>org.keyczar</groupId>
        <artifactId>keyczar</artifactId>
        <version>0.71j</version>
</dependency>
```

Here we had some difficulties because of Keyczar. It has a lot of dependencies and therefore the generated .jar was not working correctly. To solve this problem and compile the project and all the external library in one .jar file a plug in must be added in the *pom.xml* file.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven−shade−plugin</artifactId>
    <version>1.6</version>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

12.3 Problems

12.3.1 Log-in

As described in the section 11.3.1 we tried to implement the log-in functionalities with JPAM but we had some problems because, unfortunately, there are any driver for the RPi processor. After some research we found another Java binding library for PAM [59]. With this Java implementation we could, besides authenticate the user, receive additional information like the user home folder, the groups where the user belong , etc (see library documentation).

12.4 Loader

problemi con il loader che da missmatch -¿ provato a copiare libreria modificarla con nuova versione ma il missmatch rimene fino a quando non sara disponibile il repository di maven. (in questo momento sito rxtx down)

12.5 Serial communication

When we developed the serial connection in our IDE (eclipse and IntelliJ) we had no problem with the RXTX library but when we compiled directly with maven (via console) an error as occurred. The problem was that when the program was compiled with the IDE, it known the right location of the RXTX jar file and the dll files but Maven has not idea of where they were located. To try to resolve this maven dependency, we tried firstly, to install the RXTX dll files with maven [60] but this did not work. For these reasons and to have a platform independent software we decided to search for a RXTX Loader. We found a Maven dependency [61] that could detect the OS and the architecture to load the right file.

Apache common funzionano solo ogni tanto -¿ tante volte funzionava in debug ma in production non funzionava (WTF??) -¿ cercare migliore modo per sincronizzare e per scambiare errrori -¿ si ritorna codice 600 e nel codice lato client si sa bene che tipo di errore si ha quindi si stampa quello.

12.6 Application synchronization

During the development of the first command via serial connection we had some problems in the synchronisation of the two application. To solve this we decided to implement two function called *checkOK()* and *sendOK()*. The first one read a line from the console and it check if there is a 200 code that means that there are no errors. The second one write in the console the code 200 if the operation has been completed without errors. This allowed us to synchronazie the two applications.

13 Conclusion

Dire se ha senso cercare di produrre ancora dei sistemi cosi quando Zymbit sara disponibile (confronto qualita) + cosa si ha imparato in questo progetto (non fare solo quello che dice il docente ma riflettere sulle varie possibilità). Scrivere che non È HSM ma è una criptocard perchè Raspberry pi ha un proprio OS e quindi non si puo securizzare come un vero HSM (directory, programmi etc)

14 Future work

Cosa potrebbe essere ancora implementato + web/internet + sicurezza + funzionalità esterne + anti intercetto (campo elettromagnetico) + etc.

14.1 Hybrid Encryption

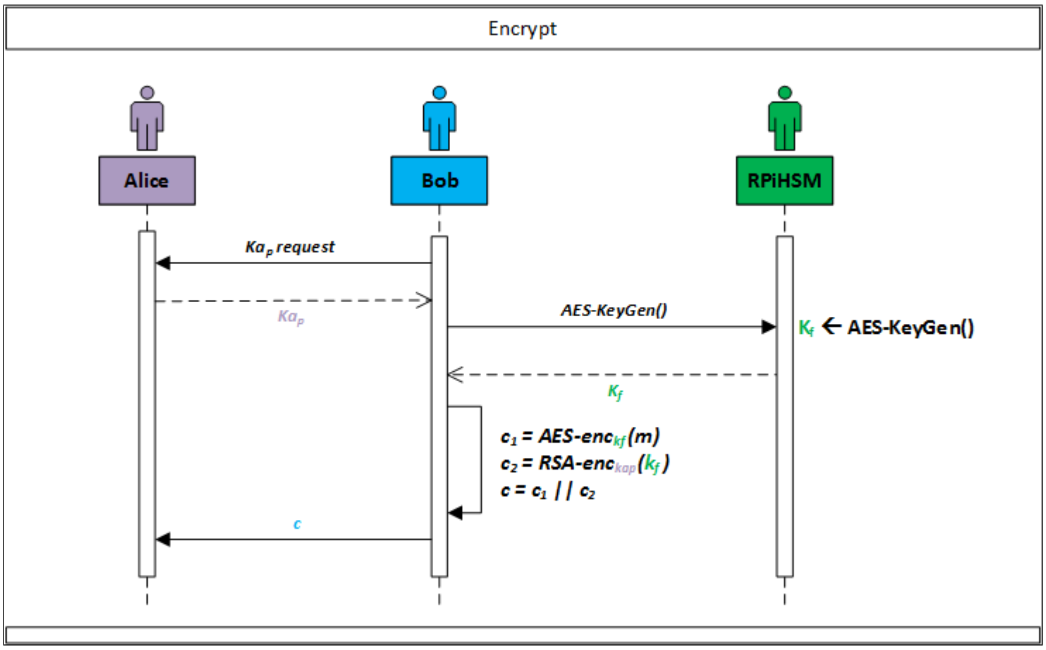


Figure 15: Hybrid Encryption Schema

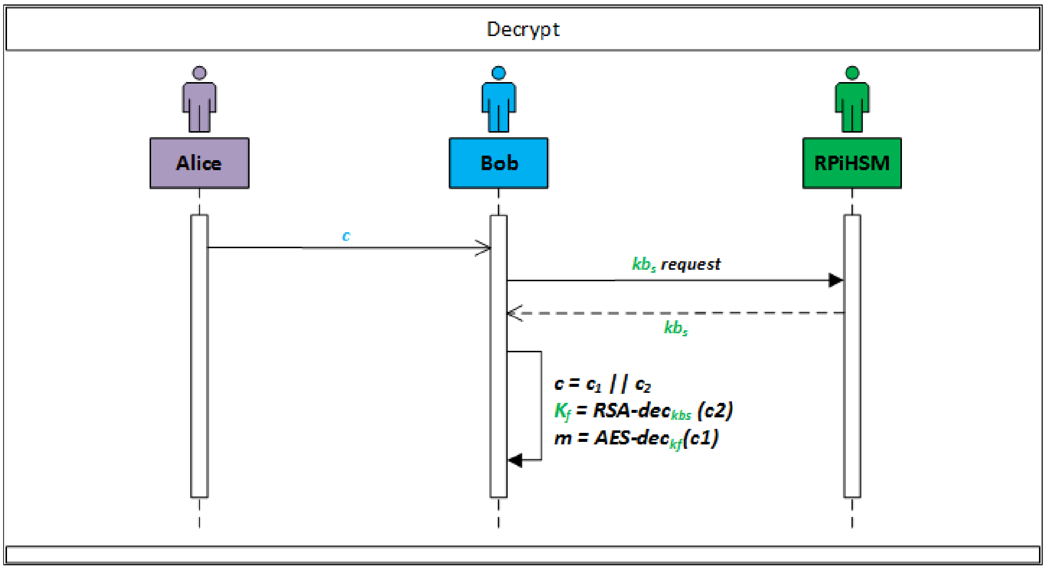


Figure 16: Hybrid Decryption Schema

15 Attachment

- Requirements Document
- Planning
- E-mails ———————— impagine emails

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] *The Trusted Platform Module explained*
`https://www.cryptomathic.com/news-events/blog/the-trusted-platform-module-explained`

[6] *Toolkit made by Google designed to make it easier and safer for developers to use cryptography in their applications.*
`https://github.com/google/keyczar`

[7] *Peter Affolter*
`https://www.ti.bfh.ch/bfh_ti/abteilungen/automobiltechnik/staff.html?ord=name&uid=27167&div=FBV`

[8] *Gerhard Hassenstein*
`https://web.ti.bfh.ch/~heg1/`

[9] *Hardware Security Modules for Protecting Embedded Systems*
`www.escrypt.com`

[10] *Advanced Encryption Standard*
`https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`

[11] *Triple Data Encryption Algorithm*
`https://en.wikipedia.org/wiki/Triple_DES`

[12] *Symmetric key block cipher*
`https://en.wikipedia.org/wiki/Camellia_(cipher)`

[13] *RSA is one of the first practical public-key cryptosystems*
`https://en.wikipedia.org/wiki/RSA_(cryptosystem)`

[14] *Description of the proof of concept done using Arduino by Stefan Arentz on his blog*
`https://stefan.arentz.ca/2012/12/30/signing-aws-requests-with-your-arduino/`

[15] *Cryptosense software detects and shows how to fix crypto vulnerabilities*
`https://cryptosense.com/`

[16] *RSA Conference is a cryptography and information security-related conference*
`https://www.rsaconference.com/`

[17] *PKCS#11 is a cryptographic token interface standard* https://en.wikipedia.org/wiki/PKCS_11

[18] *Opencryptoki open-source PKCS#11 simulator* https://sourceforge.net/projects/opencryptoki/

[19] *International standard for a secure cryptoprocessor* https://en.wikipedia.org/wiki/Trusted_Platform_Module

[20] *General-purpose input/output*

[21] *Object-oriented I/O API and implementation libraries for Java Programmers* http://pi4j.com/

[22] *Debian based OS optimized for Raspberry Pi* https://www.raspbian.org/

[23] Security Monitor for the Raspberry Pi https://www.zymbit.com/zymkey/

[24] *Crypto Shield Hookup Guide* https://learn.sparkfun.com/tutorials/crypto-shield-hookup-guide

[25] *Java ARM on BeagleBoard* http://beagleboard.org/project/java/

[26] *How to install java on Beaglebone Black* http://derekmolloy.ie/running-java-applications-on-the-beaglebone-black/

[27] *Beaglebone board Java library* https://github.com/Datenheld/Bulldog

[28] *TPM module for Beaglebone board* https://www.sparkfun.com/products/12773

[29] *The CIA triad* http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA

[30] *Infineon Technologies AG is a German semiconductor manufacturer* https://www.infineon.com/

[31] *PKI is a set of roles, policies, and procedures needed to manage digital certificates* https://en.wikipedia.org/wiki/Public_key_infrastructure

[32] *LininoOS Arduino's OS* https://github.com/Datenheld/Bulldog

[33] *RXTX Java Library to communicate over serial port between a client and Arduino* http://rxtx.qbang.org/wiki/index.php/Main_Page

[34] *JArduino Library to program and comunicate with an Arduino device* http://rxtx.qbang.org/wiki/index.php/Main_Page

[35] *Arduino crypto shield made by SparkFun* https://www.sparkfun.com/products/13183

[36] *Secure Shell (SSH) is a cryptographic network protocol*
https://en.wikipedia.org/wiki/Secure_Shell

[37] *SSH configuration guide on Raspberry Pi*
https://www.raspberrypi.org/documentation/remote-access/ssh/

[38] *Secure Copy based on the SSH protocol*
https://en.wikipedia.org/wiki/Secure_copy

[39] *Apache Maven is a software project management and comprehension tool*
https://maven.apache.org/

[40] *FTDI Cable manufacture* www.prolific.com

[41] *Tutorial to connect pc and Raspberry pi using a FTDI cable*
https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable?view=all

[42] *SSH, telnet and serial client*
http://www.putty.org/

[43] Solution to change the serial port speed to 115200
https://raspberrypi.stackexchange.com/questions/10004/how-to-start-installing-raspbian-using-serial-console

[44] Image of Learn Adafruit showing the how to connect the pins into the GPIO of Raspberry pi
https://learn.adafruit.com/assets/35695

[45] Get COM port using Devices Manager
https://learn.adafruit.com/assets/4011

[46] Putty connection over COM7 port
https://learn.adafruit.com/assets/4012

[47] FTDI Cable
https://upload.wikimedia.org/wikipedia/commons/3/37/FTDI_Cable.jpg

[48] Ethernet Cable
https://static.pexels.com/photos/257906/pexels-photo-257906.jpeg

[49] Client/Server on the Raspberry Pi
http://cs.smith.edu/dftwiki/index.php/Tutorial:_Client/Server_on_the_Raspberry_Pi

[50] RS232 Interface
https://upload.wikimedia.org/wikipedia/commons/e/ea/Serial_port.jpg

[51] Installing a RS232 Serial Port Guide
http://www.savagehomeautomation.com/projects/raspberry-pi-installing-a-rs232-serial-port.html

[52] PKCS# Java Wrapper
https://jce.iaik.tugraz.at/sic/Products/Core_Crypto_Toolkits/PKCS_11_Wrapper

[53] Serial console enable and disable script *https://github.com/lurch/rpi-serial-console/blob/master/rpi-serial-console*

[54] *A brownout is an intentional or unintentional drop in voltage in an electrical power supply system* *https://en.wikipedia.org/wiki/Brownout_(electricity)*

[55] X MODEM
http://web.mit.edu/6.115/www/amulet/xmodem.htm

[56] Python Pam *https://pypi.python.org/pypi/python-pam/*

[57] Pluggable authentication module (PAM) is a mechanism to integrate multiple low-level authentication schemes into a high-level application programming interface *http://www.linux-pam.org/*

[58] A Java-PAM bridge *http://jpam.sourceforge.net/*

[59] Java binding for libpam.so *http://libpam4j.kohsuke.org/*

[60] Stackoverflow solution to manage dll with Maven *https://stackoverflow.com/questions/1001774/managing-dll-dependencies-with-maven*

[61] RXTX loader maven dependency *https://github.com/reines/rxtx*

[62] GNU Lesser General Public License *https://www.gnu.org/licenses/lgpl-3.0.html*