AALBORG UNIVERSITY
STUDENT REPORT

ED1 - P1 - H103

# RFID based bicycle locking system

*Students:*
Antal János Monori
Emil Már Einarsson
Gustavo Smidth Buschle
Michal Damian Budzinski
Thomas Thuesen Enevoldsen

*Supervisor:*
Akbar Hussain
Torben Rosenørn

December 18, 2014

**AALBORG UNIVERSITY**

STUDENT REPORT

**School of Information and Communication Technology**
Niels Bohrs Vej 8
DK-6700 Esbjerg
http://sict.aau.dk

**Title:**
RFID based bicycle locking system

**Theme:**
Scientific Theme

**Project Period:**
Fall Semester 2014

**Project Group:**
H103

**Participant(s):**
Antal János Monori
Emil Már Einarsson
Gustavo Smidth Buschle
Michal Damian Budzinski
Thomas Thuesen Enevoldsen

**Supervisor(s):**
Akbar Hussain
Torben Rosenørn

**Copies:** 1

**Page Numbers:** 66

**Date of Completion:**
December 18, 2014

**Abstract:**

The report focuses on how bicycle sheds in Denmark can be improved to reduce theft. Bicycles are stolen from public places such as schools and train stations because they are not secured to the surroundings and can therefore lifted from their parked location. The suggested solution attempts to encourage the users to use a supplied chain-locking system that allows the users to lock their bicycles to the modified racks. The prototype uses RFID technology to associate the user with the locked bicycle. The system uses a solenoid to operate the locking mechanism and a chain that is connected to the racks. The report concludes that RFID is a viable option for securing bicycles against the given issue.

# Contents

**Preface** **1**

**1 Introduction** **2**

1.1 Initiating Problem . . . . . 2

**2 Problem Analysis** **3**

2.1 Current bicycle locking mechanisms . . . . . 5

2.2 Preliminary Research . . . . . 6

2.3 Stakeholder analysis . . . . . 7

2.3.1 Interested parties . . . . . 7

2.3.2 Actors . . . . . 8

2.3.3 Technology carriers . . . . . 8

2.3.4 Summary . . . . . 9

2.4 Risk management . . . . . 9

2.4.1 Power outage and interruption . . . . . 9

2.4.2 RFID card theft and loss . . . . . 10

2.4.3 Electrocution by contact . . . . . 10

2.4.4 Vandalism and theft . . . . . 10

2.4.5 Privacy concerns of RFID technology . . . . . 10

2.4.6 Overview . . . . . 11

2.5 Environmental impact . . . . . 13

2.6 Health and Safety . . . . . 13

2.7 Locking Mechanisms . . . . . 14

2.7.1 Technical details . . . . . 14

2.7.2 Power Outage . . . 15
2.8 Solenoid . . . 15
2.9 Economic impact . . . 16
2.9.1 Servo locks . . . 16
2.9.2 Solenoid locks . . . 17
2.10 RFID . . . 18
2.11 Relays . . . 20

**3 Problem Definition** **23**
3.1 Conclusion of Problem Analysis . . . 23
3.2 Problem Delimitation . . . 24
3.2.1 Prototype . . . 24
3.3 Requirements . . . 24
3.3.1 Prototype . . . 24
3.3.2 Testing . . . 25
3.4 Hypothesis . . . 25
3.5 Problem Definition . . . 25

**4 Development** **27**
4.1 Description . . . 27
4.2 Design . . . 28
4.2.1 Code . . . 30
4.2.2 Locking enclosure . . . 32
4.3 Specifications . . . 33
4.4 Documentation . . . 34
4.4.1 Code . . . 34
4.4.2 Build . . . 38
4.5 Testing . . . 40

**5 Discussion** **41**

**6 Conclusion** **44**

**7 Perspective** **45**

**Bibliography** **49**

**8 Appendix** **52**

8.1 Lock data . . . . . 52

8.2 Codes and header files . . . . . 53

8.2.1 Prototype code . . . . . 53

8.2.2 MRFC522 RFID reader header file for main code . . . . . 59

8.2.3 LCD $I^2C$ header file . . . . . 64

# Preface

The project entitled 'RFID based bicycle locking system' was made by five students from the Electronics and Computer Engineering programme at Aalborg Universitet Esbjerg, for the P1 project in the first semester.

From hereby on, every mention of 'we' refers to the five co-authors listed below.

Aalborg University, December 18, 2014


Antal János Monori
<amonor14@student.aau.dk>

Emil Már Einarsson
<eeinar14@student.aau.dk>

Gustavo Smidth Buschle
<gbusch14@student.aau.dk>

Michal Damian Budzinski
<mbudzi14@student.aau.dk>

Thomas Thuesen Enevoldsen
<tten14@student.aau.dk>

# Chapter 1

# Introduction

Bicycles are a popular means of travelling in several different countries around the world. In 2013, bicycles outsold passenger cars in almost every European country [1]. This popularity also requires well implemented and state-wide bicycle infrastructure, in which fortunately Denmark is one of the front-runners in Europe and around the world [2]. Parking spaces for bicycles are quite common in cities and around offices, public parks and government institutions. As bicycle sheds are freely accessible, and usually with no surveillance, it is easy to steal bicycles from within, due to the fact that the bicycles are not locked optimally.

## 1.1 Initiating Problem

Can we reduce bycicle theft by implementing a different way of parking in bicycle sheds?

# Chapter 2

# Problem Analysis

According to the Danish Bank of Statistics, there has been reported an average of 540,599 bicycle theft incidents since 2007 [3]. Bicycles are most vulnerable to theft when left at public locations such as train stations, bus stations, in front of malls or even at public schools.

Factory new bicycles usually come fitted with some sort of locking mechanism. This allows the user to lock the bicycle at any given location, the limitations of the default lock usually do not allow the bicycle to be locked to the surroundings, leaving the bicycle vulnerable to being moved or taken even though it is locked. In Denmark, organised bicycle crime is common and thieves often steal large quantities of bicycles from public locations during every hour of the day. There has been multiple reports by Danish news establishment about public schools having issues with thieves stealing the pupils bicycles. These thieves steal truckloads of locked bicycles from sheds, schools and other public places, and then drives them to a foreign country. There, they easily remove the locks and the bicycles are sold or stripped for parts [4] [5]. This means that bicycles of every price class are vulnerable to being stolen, since brand new bicycles are sold as they are and older bicycles are just stripped for parts and sold individually.

Reports have shown that this issue is preventable, but it requires more media coverage and attention. The government and other organizations throughout the world also need to be better at raising awareness on the issue of bicycle theft and should supply better designed bicycle racks, sheds and other solutions that serve as theft prevention.

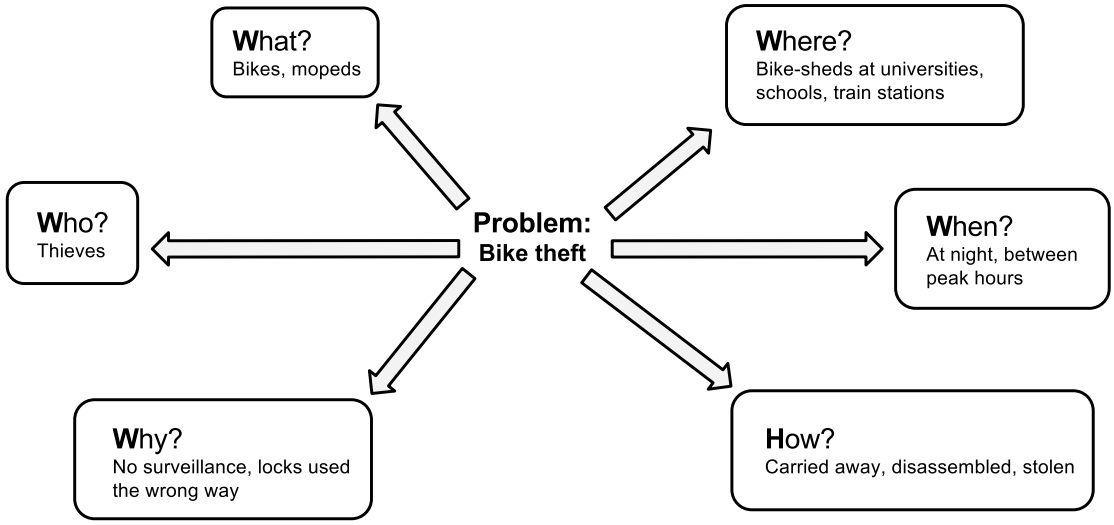


**Figure 2.1:** W-diagram of the P1 problem

To take a further look at the problem, we have created a W-diagram. This gives us an overview of the different effects that this problem causes and who are actually influenced by the main problem.

The **What** section provides information about what the problem revolves around. In this case it is bicycles and mopeds. We will be focusing on solving the issue mainly for bicycles, but the solution will also be usable by moped users.

The **Why** section of the diagram shows the cause of the main problem and our main issue to solve. Solving the **Why** part is our main focus in this report. We will take a look at the different available locking mechanisms and how they are being used.

The **How** section of the diagram indicates what the actual physical issue is and what it is there needs to solved. Based on how the bicycles are stolen, we will attempt to come up with solution to prevent bicycle theft. The main focus will be on preventing thieves from lifting the locked bicycles from their parked locations.

## 2.1 Current bicycle locking mechanisms


**Figure 2.2:** One of the most common locking mechanisms [6]

Danish bicycles nowadays already come fitted with a mounted lock from the factory. The lock seen on the figure above is one of the most widely used here in Denmark, since this kind of locking device is really convenient for the user of the bicycle. These locks are already fitted to the bike so the only thing you need to bring is the key. Alternative locking methods exist, some cyclists use chain locks that allow them to secure their bicycles to racks and other things that are part of the environment. The downsides of using a chain locking system is the inconvenience of riding the bicycle while carrying a chain lock.


**(a)** **(b)** **(c)** **(d)**

**Figure 2.3:** The most common locks available on the market. [7] [8] [9] [10]

The governmental institutions, along with many other public places, they all supply different kinds of bicycle parking solutions. Most, if not all of them, serve the purpose of keeping your bicycle upright when parked, but also to organize multiple bicycles into a smaller space. Lock (b)(c)(d) shown above allow the cyclist to lock their bicycle to the environment, but carrying one of these locks are usually an inconvenience. Therefore, most cyclists use the mounted locks (a), since these kinds of locks are not annoying or irritating to carry around.

To get an idea of how many people actually used the mounted locks(a), compared to the other type of locks, we decided to take a sample from Aalborg University Esbjerg's bicycle sheds on Tuesday the 11th of November from 9.00AM to 10.00AM.

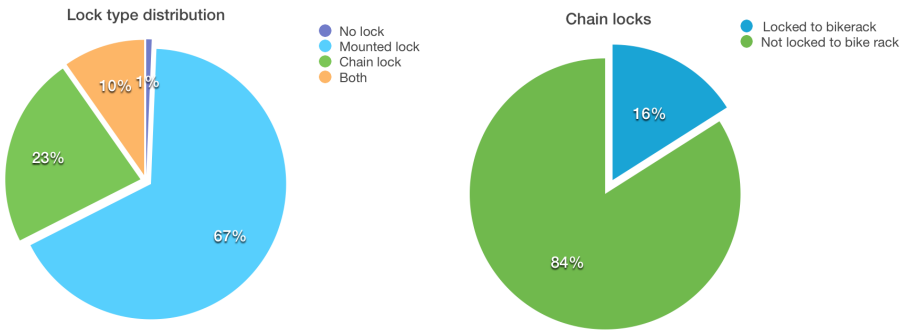


**Figure 2.4:** Sample set of data from Aalborg University Esbjerg's bicycle shed
8.1

From our sample, this set of data indicates that 67% use only the mounted locks, which removes the ability to lock the bicycle to the surroundings. Less 16% of the people who own a chain lock actually uses the lock to secure their bicycle to the bicycle rack. Even though it is a small sample it gives us a general idea on what kind of locks are in use in Denmark. To be completely certain more sampling is needed and a greater numbers of statistics must be gathered to draw conclusions.

## 2.2 Preliminary Research

Based on the statistics from the bicycle sheds at our university, we can clearly see the lack of chain locks. This makes the bicycles vulnerable to theft, where the locked bicycles are carried away from the shed since they are not locked to the racks.

Since most bicycles are equipped with a mounted lock, we are going to be looking at implementing a locking solution into the actual bicycle sheds, so that the location where the bicycle is parked becomes more secure and the bicycle user avoids having to carry unnecessary equipment. Some of the options to make the bicycle shed more secure can be listed as the following:

- A RFID activated door for the shed (Only allows authorised people)
- RFID activated chain locks
- Alarm chips in every bicycle to detect removal from the shed

Today it is quite common to carry around a card with RFID, sometimes even more than one. Therefore, by choosing a solution that uses this technology seems like a logical decision. Even if the person does not own a RFID card, it is easy to supply these kind of cards.

Every student at Aalborg University Esbjerg has a RFID student card, which students use to enter the university buildings outside of opening hours. Besides Aalborg University many other institutions use this technology, the Danish public transport also heavily relies on RFID technology. In 2003 the major Danish public transportation companies combined with one-another to create a unified system that uses a card named *Rejsekort*, which in English means "Travel card". This card uses RFID technology to keep track of how many funds are available on the traveller's card and where the person is travelling [11].

Taking a look at the first suggestion mentioned above, the addition of a RFID activated door would add some limitations to the bicycle shed. Every user of the bicycle shed must own an authorised RFID card to enter. This becomes an inconvenience if the bicycle shed is placed at a public location, since you need to be added to the whitelist to access the shed.

But instead by supplying RFID activated chain locks, it is possible for everyone to use the shed, even though they do not have a card. The user is still able to park wherever, the chain locks are supplied as optional security. So even though the user chooses not use the chain locks, the bicycle can still be parked in the rack

To conclude the preliminary research we will be taking a closer look at the different elements needed to create a RFID activated chain lock. This includes investigating the different stakeholders and risks associated the problem, but also the technology needed in creating the lock setup.

## 2.3 Stakeholder analysis

### 2.3.1 Interested parties

Interested parties are the groups of people who are being affected either negatively or positively by the product in question. They have some influence over the development, as the trends and actions of these interested parties has to be taken into consideration by the development team.

One of our main stakeholders are bicycle owners of all ages and all genders who own an RFID-based card or will be issued with one by the company who implement our solution. We expect to gather feedback from them and improve our product based on their inputs.

Companies will be the core stakeholders in this project, as they are the ones who actually implement the product for their customers. These can vary from local companies to public schools and universities to train stations and shopping malls.

Public authorities share interest in decreasing the number of bicycle-thefts, therefore will be consulted on the safety aspects and the impact of the project. These authorities might be interested in the project, if they see a potential of decreasing bicycle theft.

The general population's interest lies within the organization of bicycle-sheds around public places and the safety of their neighbourhood.

Bike thieves are considered interested and may be looked upon as competitors. It is possible to assess how they act and steal bicycles, and make our design based on those learnings. These people will not support our project in any way, as we are indirectly putting them out of business.

Companies that manufacture RFID-based equipment and solenoids would benefit from the implementation of such system, by growing their business through supplying solenoids, RFID chip based ID cards and readers.

### 2.3.2 Actors

Actors are a subset of *Interested parties*, who have any influence on the development of the project, either by financing it, providing knowledge or help the development of the project.

The supervisors involved with this project are Akbar Hussain and Torben Rosenørn. They are the people overseeing the project and who will grade and evaluate the final outcome. They also provide valuable feedback and help through supervisor meetings to Group H103 during the project.

Insurance companies would most likely be interested in investing in the idea, as it could potentially decrease the amount they need to cover for their clients. Investment funds might find the market and the product good enough that they might invest in it early stage.

### 2.3.3 Technology carriers

Technology carriers are a subset of *Actors* who have influence to change the direction of the project.

Group H103 is the group who develops the solution for this project. The group consists of:

- Antal János Monori
- Emil Már Einarsson
- Gustavo Smidth Buschle
- Michal Damian Budzinski
- Thomas Thuesen Enevoldsen

### 2.3.4 Summary

| Name of stakeholder | Type | Role |
|---|---|---|
| Bike owners | Interested parties | End-users |
| Companies | Interested parties | Clients |
| General population | Interested parties | End-users |
| Bike thieves | Interested parties | Competitors |
| RFID Manufacturers | Interested parties | Providers |
| Solenoid Manufacturers | Interested parties | Providers |
| Supervisors | Actors | Supervisors |
| Insurance companies | Actors | Possible investors |
| Public authorities | Actors | Possible investors |
| Investment firms | Actors | Possible investors |
| Group H103 | Technology carriers | Developers |

**Table 2.1:** Stakeholder table

The table above contains the names of the different people involved with or interested in the project, with the type, role and impact of their involvement specified. The type of the stakeholder represents a level of involvement within the project, meanwhile the role represents a more concrete function.

## 2.4 Risk management

Possible risks that could occur before, during and after the implementation of the proposed bicycle-locking system, elaborated in this report, and presented in the form of a table together with a mitigation plan for each and every identified risk.

### 2.4.1 Power outage and interruption

Power outage is not as common in Denmark as in many other European countries [12] due to a more developed infrastructure, therefore the risk of having power outages is low, but not necessarily impossible. Because the bicycle locking system is powered by electricity, we need to take into consideration the fact, that people may cause power outages on purpose to steal the bicycles parked in the sheds.

To mitigate this risk, we decided to go with a fail-closed design, this means that the bicycle will stay locked, rather than unlocking during a power outage. To eliminate the problem this design causes, there could be implemented a backup generator that would be used until the electricians are notified and the power is turned back on. To overcome the problem of power interruption on purpose, the power cables would also be sealed and made inaccessible.

### 2.4.2 RFID card theft and loss

People could lose their cards or someone could steal them in order to get to one of the bicycles or any other possessions that could be accessed using the card. As this risk has a very high probability, a mitigation plan needs to be put in place that could register new- and de-register old cards.

By creating an external database and providing tools to our clients allowing them to manage the registered cards in the database, we allow them to make it easier to replace cards and block them as soon as people notify them on card theft or loss.

### 2.4.3 Electrocution by contact

For safety concerns, all electronics in general should be sealed and made impossible to get access to it without proper disassembly. Proper casing should be applied to the cables and general electronics.

### 2.4.4 Vandalism and theft

There is the possibility that the product could be damaged by people both intentionally or unintentionally. It is also possible that people would disassemble the product to steal the components for their high value.

To overcome this problem, the product components should be re-evaluated for their replacement cost. For example, using gold for electric contact with the Arduino might be the best theoretical option, but the cost as a separate entity and the replacement cost is very high, due to it being a rare material. Therefore using other, more in-expensive materials could solve this problem, even if it would degrade the performance of the system to a reasonable level.

### 2.4.5 Privacy concerns of RFID technology

Some people from certain groups in society heavily oppose the widespread use of RFID technology. Some users are afraid that the implementation of RFID into our everyday lives, will void our privacy in some way. It is possible to steal someone's RFID identity without the victim knowing, and this results in the victim not being able to do anything about it. Even though there is this disadvantage with using RFID, the technology is still widely used in many products all over the world. Passports, ID badges and many other everyday items are embedded with RFID chips.

By educating our clients and end-users of our implementation of RFID technology and the information we store in our databases, it could make them more comfortable using our product, even though they might be skeptical of the technology - which is something we cannot change.

### 2.4.6 Overview

We listed most of the risks that would influence the design process of our product. These risks will be used as criteria for the end-product.

Risks are considered on two different scales with 4 levels. As the legend under the table shows, it starts from Very High (VH) being the most severe to Low (L), which can be considered as a minor issue. The two scales are **Probability**, which represents the chance of occurrence, and **Impact**, which represents the severity of the risk in case it would actually appear.

| **Risk factor** | **Probability** | **Impact** | **Mitigation strategy** |
|---|---|---|---|
| Power outage | M | H | Place a backup generator that would power the system for a while, but the fail-safe system would keep the bikes locked and safe until the power is restored. |
| Power interruption on purpose | L | VH | We seal all the cables and electronic devices from outside contact. |
| Loss of RFID card | H | M | Implement a system that can register new RFID cards and recover old ones. |
| RFID card theft | H | H | Set up a hot-line for deactivation. |
| Electrocution by contact | L | VH | Seal the electronics and limit the possibility to gain access to them without disassembling. |
| Theft of the product because of rare materials used in it (like silicon or gold) | H | VH | We evaluate the use of these rare materials and try to replace them with less valuable materials that would balance the cost of replacing the product with the price of manufacturing a new one. Instead of gold we may use a less valuable metal that also conducts electricity. |
| Vandalism | M | H | Seal the system into a wall, or create a special casing that would make it impossible to damage the internal electronics without disassembly. |
| Privacy concerns of RFID technology | L | L | We could educate users of our product by showing them how did we implement RFID technology and what information do we store and use. |

**Table 2.2:** Risk management and mitigation

**Legend: VH - Very High | H - High | M - Medium | L - Low**

## 2.5 Environmental impact

All electronic waste should be separately collected and treated according to the Waste of Electrical and Electronic Equipment (WEEE) directive 2012/19/EU [13], effective as of 14 February, 2014. To enable this, we have to make the product to be easily disassembled, without the needs of any industrial tools. This aspect needs to be taken into consideration when finalizing the industrial design and selecting the right components for the final product.

Using materials like ABS (Acrylonitrile butadiene styrene) plastic, stainless steel, copper and aluminium would qualify our product as highly recyclable.

ABS plastic is completely recyclable and very durable, therefore it is used a lot to make cases and shrouds. Recycled ABS can be blended with virgin materials to produce products with lower cost while preserving the high quality [14], therefore it is very economically attractive.

Stainless steel is also completely recyclable, and it has a long life span as well, sometimes spanning over decades [15].

Electronics containing copper are covered by the WEEE directive and is also recyclable. It is actually cheaper to re-use old- than to produce new copper [16].

Aluminium is completely recyclable and retains all of its properties even when it has been reused. Over 75% of all aluminium produced up until today is still in use [17].

For prototyping Plexiglas, ABS plastic, plywood and stainless steel is ideal for its high availability and affordable price range.

Plexiglas is a solid, durable, ecological and recyclable piece of transparent plastic that is commonly used instead of glass.

Plywood, as most wood, is recyclable and easily accessible. In a production environment, it has to follow standard codes and regulations, but it fits well for prototyping and support mounting.

Besides all this, our product should not contain any chemical substances specified in the RoHS directive 2011/65/EU [18].

These are some of the materials that we considered using and the list may change in the future, just as regulations could. Therefore a more thorough analysis needs to be taken place later on in the development of the final product, if the prototype validates the proposed idea.

## 2.6 Health and Safety

Implementing an electrical solution into a bicycle shed can pose a threat when it comes to electrical wires. It is important that the different weather conditions are kept in mind when dealing with outdoor electrical circuits, where there might

be harmful or potentially lethal electrical conditions. Rainy weather, heavy fog and general moisture are some of the weather conditions that pose a safety threat towards our electrical circuits.

All of the materials mentioned in the previous section, that were examined from an environmental standpoint, does not have any direct health concerns towards end users. For example, ABS plastic is only hazardous when heated up during production, as it may produce fumes of acrylonitrile, that is acutely toxic and highly flammable [19].

## 2.7 Locking Mechanisms

Electric locks can generally be said to have three separate behaviours during a power outage. They can either, lock, unlock, or remain at the state they were before the power outage. Locks that unlock in a power outage can be said to be fail-open, and those that lock can be said to be fail-closed, finally, those that maintain the same state can be said to be fail-constant.

The two first lock-types are best suited to two different tasks:

- Fail-open locks can be found in apartment-building doors, offices or other places that may have people located inside them. Electrical locking systems which can not be accessed and used manually, would pose a threat and safety hazard for those who are trapped inside due to malfunction.
- Fail-closed locks are used in safes and other containers for valuable items. This choice is due to the need for fewest vulnerabilities. If a power outage happens, the locks will remain closed, so there is nothing to gain from maliciously cutting the power supply to the locks.

The third type can simulate the previous two by the use of a back-up generator, or some such device. The reliability of fail-constant locks, when they simulate the other two systems, is based entirely on how this simulation is done. If, for example, the simulation is done by software, any bugs on the software might cause the lock to have undesirable behaviours.

### 2.7.1 Technical details

Electrical locks can be made with a solenoid system. This system consists of a solenoid coil, a ferromagnetic piece (armature) that moves back and forth, and a spring that moves the piece to its default position when coil is off. This system is implemented slightly differently between the three types of locks. In a fail-open system the coil pulls the piece outwards, and the spring pulls the piece inwards. In a fail closed system, the coil pushes the piece outwards, and the spring pulls it inwards. In a fail constant system there is no spring. Instead there are two coils which induce magnetic fields in different directions.

Electrical locks can also be made with servos. A signal would cause the servo to drive the armature to the desired position. Electrical locks made with servos are usually fail-constant.

### 2.7.2 Power Outage

For the fail-open and fail-closed systems, the behaviour during a power outage is simply as was described before. If the power is cut, the spring will take over, and set the armature to its default position.

For the fail-constant system, the behaviour can be made in a variety of ways. The simplest of which would be for it to simply remain where it is. However for a simulation of the previous two the system can be described as such: In the event of a power outage, a back-up device would kick in, and a signal would be sent to the locks, so that they lock or unlock, depending on the desired behaviour. This makes these locks a good low-power substitute for the previous two, but this comes at a cost of safety.

## 2.8 Solenoid

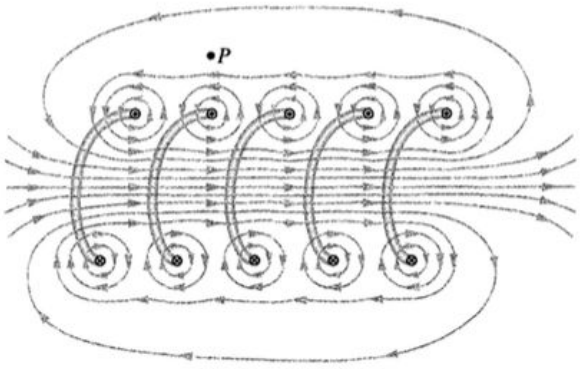


**Figure 2.5:** Solenoid magnetic field [20]

A solenoid is a loop of conducting wire. Solenoids produce a magnetic field when a current flows throught them, and vice-versa. The behaviour of a solenoid is dictated by Len's law: *"An induced electromotive force (emf) always gives rise to a current whose magnetic field opposes the original change in magnetic flux."* [21]

Given an opposite current flow, the changing magnetic field that is induced will switch polarity.

If a magnetic object is placed inside of the solenoid that has current flowing through it, the object will move according to its polarity and the direction of the current flow.

The loop usually is longer than wider, so it produces a stronger magnetic field inside. [20] A lower magnetic field outside of the loop and higher inside creates a force towards the center of the coil. This causes the pin inside of the solenoid to center itself.

These characteristics are widely used in electromagnetic (electromechanical) locks.

## 2.9 Economic impact

This section will examine the running costs of our concept, and will be used to examine the different solutions to our locking mechanism. These numbers are based on rough calculations as of 26th of November, 2014 and all numbers are represented in Danish krones.

In order to calculate costs, we need to assume that the locks will need to be powered approximatively two-to-four times for 1-10 seconds. Electricity prices for households in Denmark, as of January 1st of 2013, is 2.26 DKK per kWh with an annual consumption of 4000 kWh [22]. We will compare the running cost of a servo lock and a solenoid lock. For each locks we are going to look at the cost of running for a day, a week and a year.

### 2.9.1 Servo locks

For servo locks, we decided to take a closer look at two servo mechanisms; the RF-020TH from Mabuchi Motos [23] and HS-805BB from Hitec [24]. The RF-020TH has a an operating voltage of 4.5 V. The electronic current in the condition where there is no load is 58 mA, while it has load it is 184 mA, with an operating voltage of 5 V. The HS-805BB has 8 mA while having being idle, and 700 mA under load. We presume that the servo locks need to be powered four times (the servo needs power to move in both forward and backwards position) for 10 seconds.

To calculate the running costs on DC, we use the following formula to calculate the Watt usage first:
$\mathbf{P}_{(W)} = \mathbf{I}_{(A)}\ \mathbf{x}\ \mathbf{V}_{(V)}$

To translate this for our to servo locks, it looks like the following:
**RF-020TH with no load:** 0.058 A x 4.5 V = 0.261 W
**RF-020TH with load:** 0.184 A x 4.5 V = 0.828 W
**HS-805BB with no load:** 0.008 A x 5.0 V = 0.040 W
**HS-805BB with load:** 0.700 A x 5.0 V = 3.5 W

Using the watts and the time factor, we can easily calculate the Wh usage:
$\mathbf{E}_{(Wh)} = \mathbf{P}_{(W)}\ \mathbf{x}\ \mathbf{t}_{(hr)}$
We also have to note, as we estimated, that the servo mechanisms will be used 4

times in a day, that means it will be idle 23.988889 hr and under load for 0.011111 hr in a day.

To translate this for our to servo locks, it looks like the following:
**RF-020TH with no load:** 0.261 W x 23.988889 hr = 6.261100029 Wh
**RF-020TH with load:** 0.828 W x 0.011111 hr = 0.009199908 Wh
**HS-805BB with no load:** 0.040 W x 23.988889 hr = 0.95955556 Wh
**HS-805BB with load:** 3.5 W x 0.011111 hr = 0.03888885 Wh

To sum up the kWh usage for the models, we add the two conditions together and divide by a 1000:
**RF-020TH:** (6.261100029 + 0.009199908) / 1000 = 0.006270299937 kWh
**HS-805BB:** (0.9595555600 + 0.038888850) / 1000 = 0.00099844441 kWh

Now, knowing the kWh price in the local currency, we calculate the running cost for both models with four digit precision:
**RF-020TH:** 2.26 DKK x 0.006270299937 kWh = 0.0142 DKK
**HS-805BB:** 2.26 DKK x 0.00099844441 kWh = 0.00256 DKK

This would result in the following running prices:

| Model | Running cost | | |
|---|---|---|---|
| | **1 day** | **1 week** | **1 year** |
| RF-020TH | 0.0142 DKK | 0.09920 DKK | 5.1759 DKK |
| HS-805BB | 0.00256 DKK | 0.01580 DKK | 0.8142 DKK |

**Table 2.3:** Running cost calculations for servo mechanisms

### 2.9.2 Solenoid locks

For solenoid locks, we decided to take a closer look at Lock-style solenoid by Adafruit [25] which has an operating range of 9 to 12 V. At 9 V, the electronic current is 500 mA, while at 12 V it is 650 mA. This only needs to be powered two times (the solenoid only needs power to open. a spring will move it back to its original position) a day for 10 seconds.

Following the steps we established before, we calculate the Wh usage. Again, we note that the solenoid mechanism will be used 2 times a day, meaning it will only run for 0.005556 hr per day.

**Running at 9V:** (0.500 A x 9 V) x 0.005556 hr / 1000 = 0.000025002 kWh/day
**Running at 12V:** (0.650 A x 12 V) x 0.005556 hr / 1000 = 0.0000433368 kWh/day

This would result in the following running prices:

| Operating range | Running cost | | |
|---|---|---|---|
| | 1 day | 1 week | 1 year |
| 9 V | 0.000057 DKK | 0.000400 DKK | 0.020568 DKK |
| 12 V | 0.000098 DKK | 0.000686 DKK | 0.035651 DKK |

**Table 2.4:** Running cost calculations for solenoid mechanisms

## 2.10 RFID

RFID stands for **R**adio **F**requency **ID**entification. The basic principle of communication, between the reader and the so called tag, is happening via an electro magnetic field that is emitted from a coil connected to the reader device. The tag also has a coil that picks up the electro magnetic field which generates a current in the circuit of the tag. That current is used both to power the tag as well as transmit information from the tag [26].

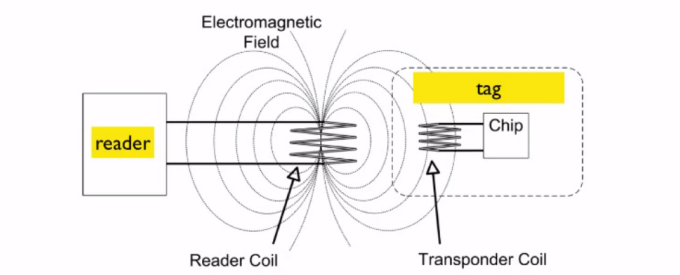


**Figure 2.6:** Electromagnetic field between reader and tag [27]

The interaction between the electro magnetic field in the reader and the current generated in the tag makes for a two-way communication. Once when the wave is directly received and the current powers the tag and the information can be extracted directly. Alternatively since a current is being induced in the antenna of the tag this current itself emits a electro magnetic field. That electro magnetic field can be received by the reader. This is called a backscattered signal. By modulating the backscattered signal we can transmit digital information. The practical way to do this is to change the load that is connected to the antenna on the actual tag. By using a switch, for example a transistor, that connects the antenna to the ground and disconnects it makes it possible for us to transmit ones and zeros [26].

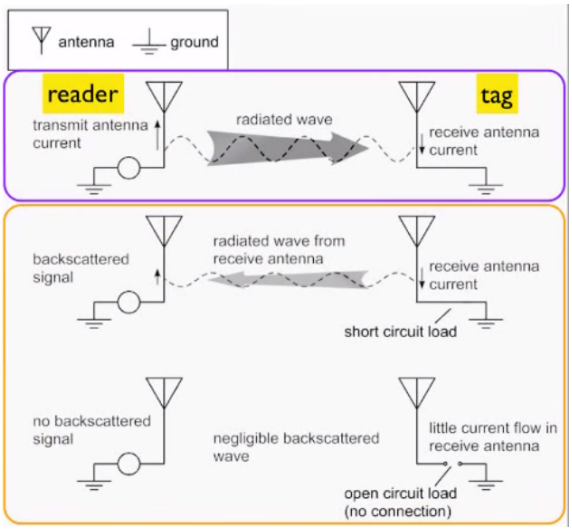


**Figure 2.7:** Backscattered data from tag to reader [27]

The reader emits a carrier wave that is (amplitude) modulated with the transmitted data. This enables the possibilities to have a constant electro magnetic field impinging on the antenna of the tag. The tag is being powered by the reader's electro magnetic wave so we cannot turn of the reader to listen because the tag could then run out of power. The signal is modulated between ones and zeros with the amplitude, the smaller amplitude contains still enough power to drive the tag. Once the current is induced in the antenna we have essentially two circuits. One takes care of the power for the circuit. There we have a diode that the current goes trough and a big capacitor that is connected to ground. If we make that capacitor big enough then the incoming RF wave after rectification is then turned into more or less constant DC bias. At the same time we have a second circuit to extract data from the reader by using envelope detector. It is essentially the same circuit except we use a much smaller capacitor so that the output signal after the diode adjusts quickly with the amplitude of the incoming electro magnetic wave [26].

If the circuit is supposed to talk back to the reader for example giving the name of the tag owner or how much money is on it, then the tag needs to send that signal back. You can use a transistor or a MOSFET that connects or disconnects the antenna from ground. By doing that the current in the antenna changes strongly and that changes the electro magnetic signal made by the tag antenna. That backscatter signal can then be picked up and evaluated by the reader [26].

Both those things are happening at the same time. This creates the issue that a strong EM signal is coming in when a much weaker signal is being emitted back. The reader device needs the be able to distinguish between those two signals [26].

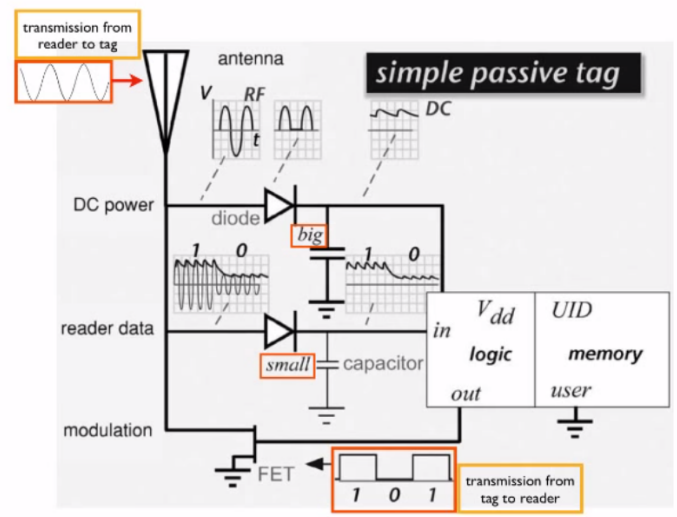


**Figure 2.8:** Power circuit of a passive tag [27]

The Mifare RFID protocol uses a passive card setup and the power comes from the reader for the tag. The cards have a 1 kb encryptable memory if it is needed to go into more security. The cards also have a processing unit than can do arithmetic operations (that could change the amount of money stored on a card, not used by us). The operating frequency is 13.56 MHz. There are several standards for RFID cards that are built on different frequencies depending on the application. The Milfare RFID protocol is optimized for a short range communication, around 100 mm [26].

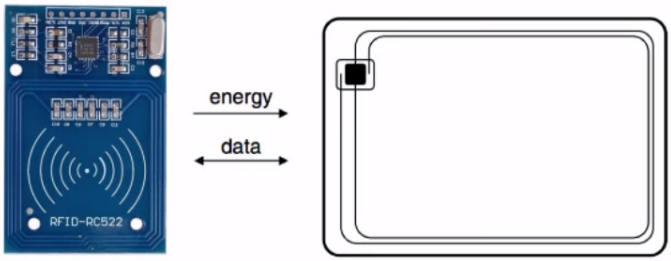


**Figure 2.9:** Milfare RFID reader and a standard RFID tag [27]

## 2.11 Relays

An electrical relay is an electrically powered switch, used when we need to electrically separate circuits, switch between more than one circuit, or manipulate a high powered circuit using a low-powered signal.

When it comes to relays, there are 4 basic types. They are the following:

**Normally Closed (NC)**
Also called a SPST (Single Pin, Single throw)

**Normally Open (NO)**
Also called a SPST (Single Pin, Single throw)

**Change over**
Single Pin Double throw (SPDT)

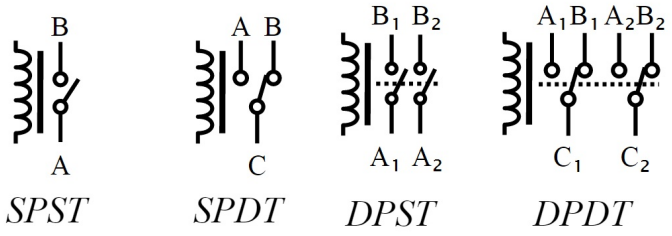


**Figure 2.10:** Basic relay overview [28]

Relays that are called SPST have 4 pins. Two pins for the coil and two terminal pins. This state of the terminal pins can either be NO or NC. When the coil is powered, the terminal pin changes position, so that the NO relay closes and the NC relay opens. Other relay types such as SPDT have 5 pins. Two pins for the coil and three terminal pins. The SPDT is basically a combination of a NO and a NC relay. Two of the three terminal pins are NO and the last one is NC. When the coil is powered, the position is swapped and the the NO becomes closed and the NC becomes open. [29]

Electrical relays can be also divided into two groups based on construction: electromechanical and electrical (containing relays built on transistors, thyristors, triacs).

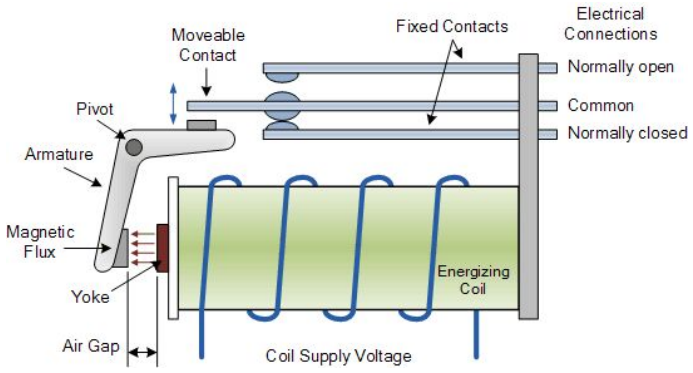


**Figure 2.11:** Electromechanical Relay [29]

Simple electromechanical (also called as electromagnetic) relays consist of an electromagnet, levers and sets of movable contacts as shown in Figure 2.11.

Electromechanical relays are inexpensive and they allow use of low power input. Unfortunately as it is a semi-mechanical device response time is high, and this changes over time due to aging effects on the moving parts.

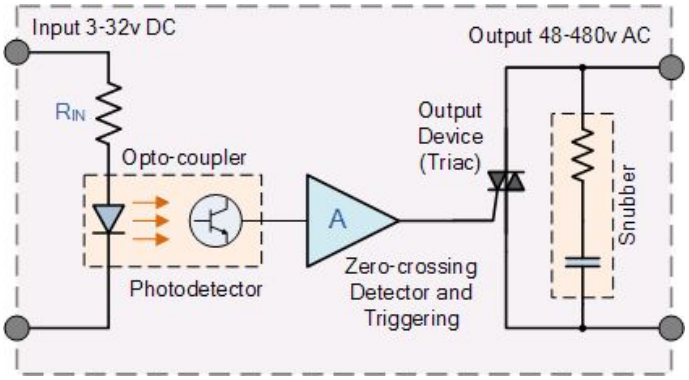


**Figure 2.12:** Solid State Relay [29]

Other widely used type of relays are Solid State Relay (SSR), built purely out of electrical components. The SSR provides high reliability, lower response time, lower electromagnetic interference and long life.

# Chapter 3

# Problem Definition

## 3.1 Conclusion of Problem Analysis

During the research for this project, we have figured out which technologies we will use when solving our problem statement. Through research we have looked at the different stakeholders and also taken into account the different health and safety measures and the environmental impact. We will be using RFID to monitor and keep track of the different users of the locking system, for this project we will be using passive RFID, since this is the technology used in our student cards and other forms of ID cards. This will make it easy for us to integrate this system into working environments, where some sort of ID system already is in place. We gathered some data from the bicycle sheds at Aalborg University Esbjerg, and seen that the mounted locks are the most commonly used locks in our sample. Besides seeing how popular the mounted lock is, we have also seen how the chain locks are used in an ineffective manner. The reason for this is that most of the bicycles are not equipped with locks able to secure themselves to the surroundings, since the most common type is the mounted lock.

The processing unit of our locking system will be a micro-controller, this micro-controller will handle the different inputs and outputs from the various components. To create a working locking solution that will work in practice we will be using a 12V or 24V solenoid. The solenoid will be the electrical component that moves a pin, so that the lock is electro-mechanically locked and unlock. Micro-controllers do not output a high enough voltage and a current to drive a solenoid, therefore we will need to work around this by using a relay. The relay will be used as an electrical switch controlled by the micro-controller to turn the solenoid on and off.

## 3.2 Problem Delimitation

Based on previous discussions, the following pieces have been selected:

**Fail-constant lock:**
Because it gives more control over the behaviour of the lock, and the unreliability is not a big issue.

**Normally closed relay:**
Since the solenoid does not require constant power.

**Short range RFID reader:**
Long range does not give any significant advantages in this case.

### 3.2.1 Prototype

The main limiting factor for us is time. In the time frame of this project we cannot build a full scope product, finding all the parts, ordering, and assembling them completely would take too long. Furthermore we do not have the tools to build a full scope product either. Hence we will build a prototype as a proof of concept.

For our prototype, we decided to use a one fail-closed solenoid as opposed to a fail-constant locking mechanism. Fail-closed solenoids are easier to operate, cheaper, and easy to find, but they are not as flexible. We will implement this locking mechanism on a 1:1 scaled bicycle rack. This prototype will not have all the features we wish to have for the final product, but the the goal is to show the interaction between an RFID reader and a locking mechanism, through a micro-controller.

## 3.3 Requirements

This section outlines the requirements for both the prototype that we are aiming to achieve through this project, and the requirements for testing and evaluating it.

### 3.3.1 Prototype

1. The model should be 1:1.
2. It should operate sequentially.
3. It should be operated by one person at a time.
4. One micro-controller should only handle one lock.
5. The system should include an RFID reader.
6. It should be compatible with existing RFID cards.

7. It should be using an Arduino single board micro-controller.
8. The program should be written in C++ and C.
9. The case for the lock and the lock should be made out of wood for easy prototyping.
10. The lock should be stripped to the rack.
11. The controller case should be pre-built.
12. A display should be used to interact with the user.

### 3.3.2 Testing

1. The lock fits in well into the locking case.
2. The touch sensor detects the lock.
3. The screen will show the right text on each step.
4. Only one card is associated with the lock at a time.
5. Our student cards can operate the device.
6. The controller correctly detects the RFID-enabled card.

These criteria will be used to design, implement and test our implementation of the prototype later on. These requirements will be re-evaluated in the Discussion section.

## 3.4 Hypothesis

Due to the fact that our report deals with such a broad scope, we can not answer our initiating problem in such a short time. Therefore a hypothesis is used to measure the success of our project.

The solution to the initiating problem implemented in this project, will make it harder for thieves to remove bikes from bicycle sheds due to inefficient locking by their owners. The solution in place will use RFID-based cards to lock their bicycles to the rack to reduce time spent on parking and reducing the stress level by preventing bicycle theft.

## 3.5 Problem Definition

The scope of this project is to create a working and affordable prototype of the above delimited problem, that would showcase the main capabilities of the end product,

and would validate the idea of solving the presented problem in the Problem Analysis chapter.

Through testing our hypothesis and by fulfilling the requirements displayed above, we will be able to determine whether or not the project is a success.

# Chapter 4

# Development

## 4.1 Description

The goal of development is to create a full-size prototype that can be used by a single bicycle. The mechanical part of the prototype will be accomplished by modifying a bicycle rack, so that a chain can secure the bicycle by its frame to the rack using an electro-mechanical lock. The electronics part uses an RFID reader to gather data from a card. An Arduino is used as the core processing unit, which will link a card with a specific lock, or to unlock a lock that is associated with the card. To ensure that the lock is in a closed position, a push-button inside of the locking enclosure checks for the locking pin. The unlocking signal from the Arduino is sent to a relay to drive the solenoid. The user of the locking system will be instructed by an LCD screen, which will display the different steps needed to lock and unlock the bicycle. The system is powered by an external power supply of 5V for the Arduino board and 12V for the solenoid lock unit.

## 4.2 Design

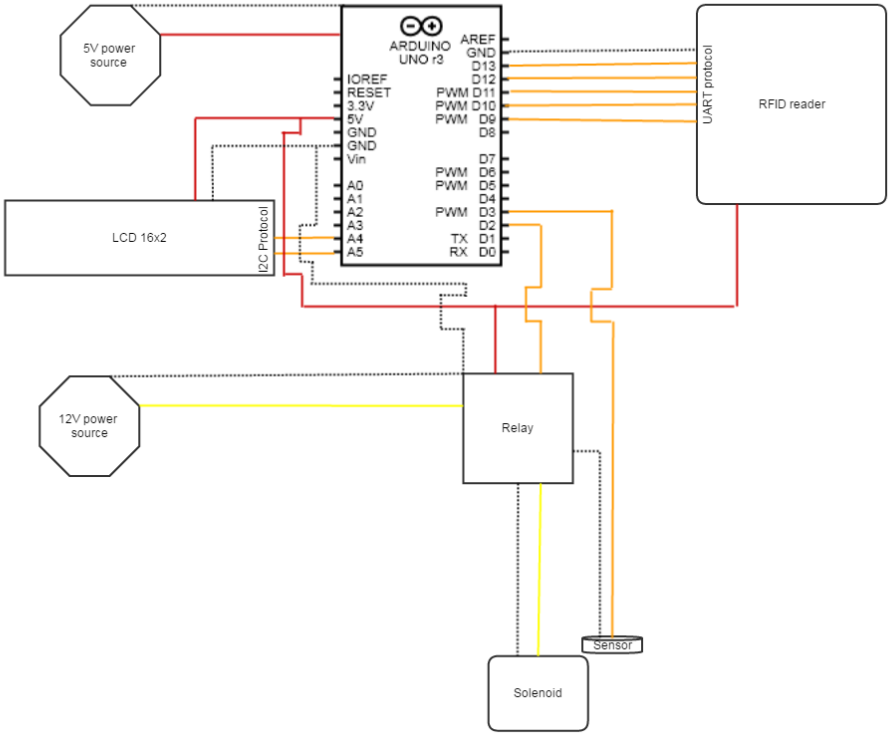


**Figure 4.1:** Full-scope picture of the design

The full-scope picture shows all connections in our design. The black dotted line is ground, yellow is 12V DC and red is 5V DC. Orange is used for data transfer. The 5V from the Arduino and D2 control the relay that opens for the 12V to the solenoid. There is a sensor in the lock that tells us when it is closed. That sends a signal to the Arduino in port D3. Ports A5 and A4 are connected to the LCD screen and it uses the $I^2C$ protocol. Ports D9-D13 connects to the RFID reader using the UART protocol (Universal asynchronous receiver/transmitter).

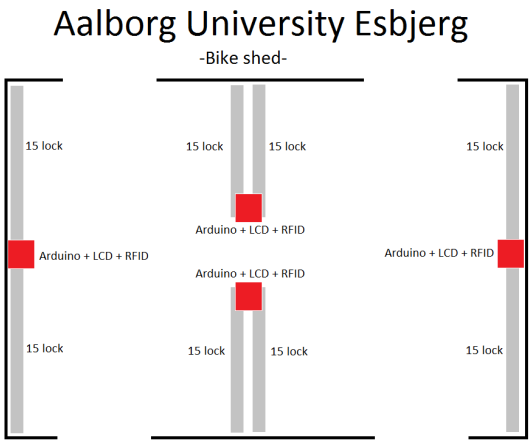


**Figure 4.2:** Aalborg University bike shed

Here is a basic concept idea of how we would implement the final design here in Aalborg University Esbjerg. For the final design a larger micro-controller would be necessary, like an Arduino MEGA, each MEGA could control up to 30 locks. All of the locks are controlled by a single RFID reader and one LCD screen. We would have our system on every other bicycle rack. So people could choose whether or not they wanted to use our system or their own locks.

### 4.2.1 Code

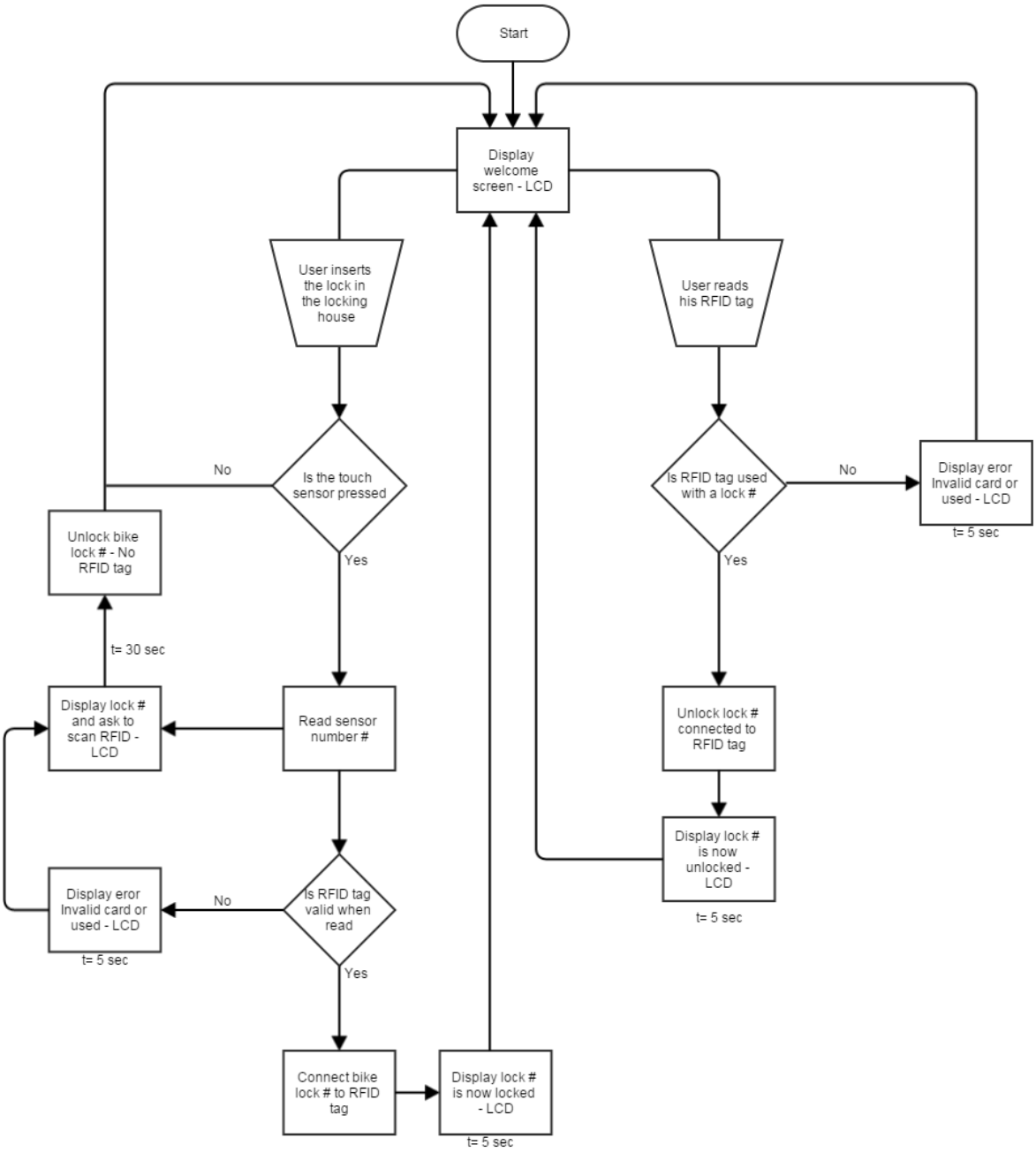


**Figure 4.3:** Flowchart of the code used

The flowchart shows a general idea of how our code will work. It will have two main functions. One is for locking new bicycles to the rack and the other is to unlock the bicycle. On the left side of the flowchart you can see a timer set t=30. If 30 seconds pass and no new RFID card is scanned to be associated with a lock, then the lock will automatically unlock. If a RFID card is read it will ignore to auto-unlock and continue with the program.

**Algorithm 1** RFID bike lock

```
LCD display - Welcome screen

while touch sensor pressed = 1 do
  sensor X = lock X
  LCD display - Is your lock number X? Then please read RFID card.

  if RFID tag is valid then
    lock X connects to RFID tag
    timer = 0
    LCD display - Bike lock X is now locked. Thank you
    touch sensor pressed = 0
  else if RFID tag is invalid then
    unlock bike lock X
    LCD display - ERROR invalid or used card, Try again.
  end if

  if timer = 30000 then
    unlock bike lock X
    LCD display - Bike lock X was not locked. Try again.
  else
    timer = timer +1
  end if

  delay 1
end while

if RFID read then
  if RFID tag used then
    unlock bike lock X that is connected to that card
    LCD Display - Bike lock number X is now unlocked. Thank you.
  else
    LCD display - ERROR invalid or used card, Try again.
  end if
end if
```

### 4.2.2 Locking enclosure

The main characteristics of the prototype lock design are:

1. Solenoid based
2. Switch as a locking sensor
3. Easy to build wooden case
4. Additional space to fit relay and cables
5. Covered by Plexiglas

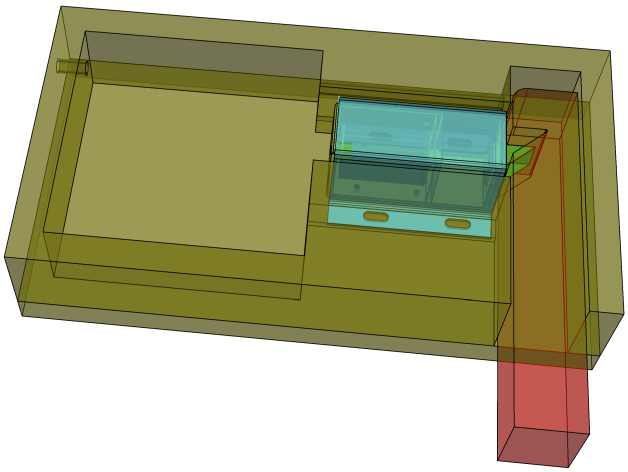

**Figure 4.4:** CAD drawing of the lock design

During the design phase we assumed that this lock would be mounted directly on a bicycle rack. The red plug connects to a chain, which is secured to the rack, allowing a bicycle to be locked to it. When the lock is unlocked, the red plug will fall out of the enclosure from gravitational force. This design feature allow the plug to be kept in place while the user is scanning their card. The end of the plug is rounded to make it easier to insert the plug into the locking enclosure.

## 4.3 Specifications

Components:

**RFID Reader:**
An RFID reader that can read RFID-tagged cards. 5.3 dollars. [30]

**Solenoid:**
A solenoid, similar to a door strike. It takes 9-12V. 15 dollars. [31]

**Micro-controler:**
Arduino UNO. Takes 5V-20V. 17 pounds. [32]

**Relay:**
Used to control the solenoid. [**?**]

**Switch:**
A small contact that closes whenever the lock-head is inside the case. [33]

**LCD Screen:**
A 16x2 LCD screen that relays information to the user. 24.4 pounds. [34]

**Chain:**
A 1.5m steel chain. Each link is 1.5cm wide, and 3mm thick.

**Lock Head:**
A simple lock head made out of plywood. It locks into the wooden casing. Built by us. 2x2x16cm

**Wooden Casing:**
A simple casing made out of plywood. Holds the solenoid and relay. Built by us. 10x18x4cm

**Electronics box:**
A box that holds the micro-controller, LCD, and RFID reader. 17x12x5.5cm

Overall, the electronics box and wooden casing are mounted on a metre high, half-metre wide plywood panel, which is attached to a metre long bike rack.

## 4.4 Documentation

The development of our product started out with early-prototyping of the code and the individual components. Right after the research and initial analysis was completed, we started looking into running the individual components and searching for libraries that we needed to use. This helped us to identify problems like lack of documentation of LCD libraries for the Arduino MEGA board, which we considered to use at some point of our research.

These smaller component-prototypes eventually became part of our prototype code.

The code for the prototype only works for the desired single-lock prototype. It can take any student card and associate it with a lock. But it will only store the card information when the lock is locked and delete the information when its unlocked. For future work we would use a centralized database to deal with multiple locks and cards.

### 4.4.1 Code

```
1  void loop () {
2      successRead = getID(); // sets successRead to 1 when we get read
       from reader otherwise 0
3      sensorCheck = sensorRead(); //Read all sensors if new sensor read
       then enter locking mode
4      welcomeScreen();
5
6    if ( successRead ) {
7      if ( findID(readCard) ) {        // If not, see if the card is in
       the EEPROM
8        openLock(500, 0);
9        deleteID(readCard);
10       // Open the door lock for 5000 ms
11     }
12     else {        // If not, show that the ID was not valid
13       failed();
14     }
15
16   }
17
18   if ( sensorCheck ) {
19     askRFID();
20     for (int i = 0; i <= 600; i++) {
21       if (i < 600){
22         successRead = getID();
23         if (successRead) {
24           notLocked=0;
25           break;
26         }
27       }
28       else {
29         failedLock();
```

```
30            openLock(500, sensorCheck);
31          }
32          delay(50);
33        }
34
35        if (notLocked == 0){
36          writeID(readCard);
37          lockLock(sensorCheck);
38          delay(3000);
39
40          lcdDelay2=0;
41        }
42        else {
43          lcdDelay2=0;
44        }
45      }
46  }
```

Prototype main loop

The main loop in the prototype code always checks if there is a new RFID tag read, or if a lock recently has been locked. If nothing is happening it will show the welcome screen. The main loop then has two main if statements. The first if statement checks if there is a RFID card read by the system and the second if statement checks if the sensor in the enclosure has been activated from the locking-peg.

If an RFID tag is read, it will check if it is a valid and connect it to a lock. If the card already is connected to a lock, the display will show a message to the user that the lock is unlocked and then unlocks the bicycle.

If the sensor in the enclosure registers a new lock it will ask the user to scan their RFID tag/card. It will check if the card is valid or connected to another lock. It will do this for 30 seconds. If no valid RFID card is read in 30 seconds it will unlock the lock and display it to the user. The system has this feature in place to avoid people locking all the locks without associating them with a RFID card of some sort.

If a valid one is read it will display a message to the user that the system is locked and that their card is associated with that specific lock.

The main loop also addresses input errors. One is if at any time an RFID card is read that is invalid for this system and another is if the user is trying to lock the second bike lock with the same RFID card it will display an error message to the user.

```
////////////////////////// Get PICC's UID ////////////////////////

int sensorRead(){
    //This goes over all the sensors to see a change in HIGH and LOW
    if (lockCheck[0] == 0) { //if a sensor has a high signal but has
   been connectd to a RFID tag then it's ignored
      val = digitalRead(3);
      if (val == 1) return 1;
      else return 0;
    }
    return 0;
}
```

Prototype sensorRead function

The function *sensorRead*() checks if the lock has been locked. The end of the lock has a sensor and it will notify the program what lock has been locked. The program then goes to check for a new RFID tag for 30 seconds. If no RFID tag is scanned it will open the lock again. If a new RFID tag is entered it will change the number in the array *lockCheck*[] from 0 to 1. So the program will ignore a previously locked lock.

```
////////////////// Lock bike ////////////////////////////

void lockLock( int locksensor ){
  //Message to user if he has used a RFID to lock a new lock
  lcd.clear();
  lcd.home();                    // go home)
  lcd.print("1 Is now locked");
  lcd.setCursor ( 0, 1 );        // go to the 2nd line
  lcd.print("   Thank you");

  lcdDelay2 = 0;

  lockCheck[0]=1;

}
```

Prototype lockLock function

Te function *lockLock*() displays a message to the user that the lock is now locked and changes the array *lockCheck*[0] = 0 to *lockCheck*[0] = 1. That is done to prevent the system to check multiple times on the same lock that has been connected to an RFID tag.

```
////////////////////// Get PICC's UID ////////////////////////////////
int getID() {
  // Getting ready for Reading PICCs
  if ( ! mfrc522.PICC_IsNewCardPresent()) { //If a new PICC placed to
   RFID reader continue
    return 0;
  }
  if ( ! mfrc522.PICC_ReadCardSerial()) { //Since a PICC placed get
   Serial and continue
```

```
8       return 0;
9     }
10    // There are Mifare PICCs which have 4 byte or 7 byte UID care if you
        use 7 byte PICC
11    //Serial.println("Scanned PICC's UID:");
12    for (int i = 0; i < mfrc522.uid.size; i++) {  // for size of uid.size
        write uid.uidByte to readCard
13      readCard[i] = mfrc522.uid.uidByte[i];
14    //  Serial.print(readCard[i], HEX);
15    }
16    //Serial.println("");
17    mfrc522.PICC_HaltA(); // Stop reading
18    return 1;
19  }
```

Prototype getID function

The function *getID*() checks if a new RFID tag is read by the RFID reader. If so it checks if the card is of the right type. It then writes it to the array *readCard*[]. It only stores 4 bytes of information, which is the ID of the card. The program ignores all of the other user data. In future versions we can hash the information, hashing is encrypting the data so that even if someone gets the data it is unusable.

```
1  ////////////////////// Check Bytes   ///////////////////////
2  boolean checkTwo ( byte a[], byte b[] ) {
3    if ( a[0] != NULL ) // Make sure there is something in the array
       first
4      match = true; // Assume they match at first
5    for ( int k = 0; k < 4; k++ ) { // Loop 4 times
6      if ( a[k] != b[k] ) // IF a != b then set match = false, one fails,
       all fail
7        match = false;
8    }
9    if ( match ) { // Check to see if if match is still true
10     return true; // Return true
11   }
12   else  {
13     return false; // Return false
14   }
15 }
```

Prototype checkTwo function

The function *checkTwo*() compares two arrays together to see if they have the same data. The function compares the *storedCard*[] array to *readCard*[], to see if the card that is read is in the system or not. For future systems where there is more than one lock, it is also necessary to compare the pointer array *lock*[] to *readCard*[] and then open the connected lock to that RFID tag.

```
1 ////////////// Check readCard IF is masterCard   ///////////////////////
2 // Check to see if the ID passed is the master programing card
3 boolean isMaster( byte test[] ) {
4   if ( checkTwo( test, masterCard ) )
5     return true;
6   else
7     return false;
8 }
```

Prototype isMaster function

The function *isMaster* only checks if the RFID tag that was read is the master card and uses the *checkTwo*[] function to do that. Master card is used by the main user to help with the system if an error comes up or a card is lost.

### 4.4.2 Build

The prototype case for the locking device is primarily built using plywood, a 12V solenoid, a mechanical relay, a switch and a Plexiglas cover.

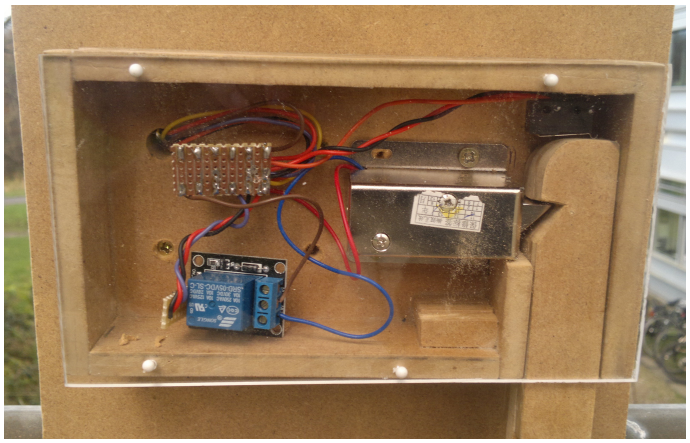

**Figure 4.5:** Picture of lock

The main prototype box contain an Arduino Uno development board (containing Atmel micro-comtroller), a RFID reader and a LCD 16x2 screen. They are placed in a ABS electrical box in which we have cut hole fitting the display. A RFID reader is located on the other side of the box, inside, under the label.

Both the locking enclosure and the case containing the micro-controller and electronic components are secured to a plywood wall and is mounted to the bicycle rack. The box containing the electronics is mounted on an extension that is tilted at an 45 ° angle, making the display text more readable. The case containing the locking mechanism is mounted directly on the wall.

**Figure 4.6:** Picture of components

**Figure 4.7:** Picture of prototype rack

The components in the main box are connected by plug-in cables and soldered where needed, according to our design. The lock case and the wall is made of plywood using basic tools. The lock case construction is glued and nailed together, built after a slightly altered design, due to the fact that more advanced tools were not available.

## 4.5 Testing

Things needed to be tested for the prototype:

1. The lock fits in well into the locking case.
   - The first test was to check the length of the chain and how well the lock fits into the locking enclosure.

2. The touch sensor detects the lock.
   - The locking device and the plug worked well together and the sensor correctly indicated that the plug had been inserted into the enclosure.

3. The screen will show the right text on each step.
   - The LCD starts with scrolling a welcome message.
   - The LCD indicates an error message to unsaved cards.
   - The LCD displays a message when a plug is put in the locking case and asks for an RFID.
   - The LCD displays a message when an RFID tag has been saved and tells the user that the lock is locked.
   - The LCD displays a message when the user scans an RFID tag and opens the lock.

4. Only one card is associated with the lock at a time.
   - When a non-saved card is used by the system in gives and error message.
   - The system only saved a card when a lock is locked and deletes it when it's opened.

5. Our student cards can operate the device.
   - We tested our Aalborg University's student cards and they worked.

6. The controller correctly detects the RFID-enabled card.
   - We tested 4 different types of RFID cards that we had on hand. Two came with the RFID reader. One was a danish travel card and the last one a AAU student card.

# Chapter 5

# Discussion

The main purpose of this discussion is to take a look at our success criteria for our prototype and our hypothesis, to discuss the different results from our development and testing. In the problem definition, we created a hypothesis and defined a certain list of prototype and testing requirements. If these points are met, the prototype will be viewed as a success.

Our hypothesis is the following: *The solution to the initiating problem implemented in this project, will make it harder for thieves to remove bikes from bicycle sheds due to inefficient locking by their owners. The solution in place will use RFID-based cards to lock their bicycles to the rack to reduce time spent on parking and reducing the stress level by preventing bicycle theft.*

To ensure that the hypothesis was thoroughly tested, we set up the prototype and testing requirements accordingly, so that we were sure that the different success criteria were met. We constructed a full-size prototype with a single lock, to get a realistic view of the design. This was to make sure that our suggested solution actually is feasible.

Below you will find a with list the prototype requirements.

1. The model should be 1:1.
2. It should operate sequentially.
3. It should be operated by one person at a time.
4. One micro-controller should only handle one lock.
5. The system should include an RFID reader.
6. It should be compatible with existing RFID cards.
7. It should be using an Arduino single board micro-controller.
8. The program should be written in C++ and C.
9. The case for the lock and the lock should be made out of wood for easy prototyping.

10. The lock should be stripped to the rack.
11. The controller case should be pre-built.
12. A display should be used to interact with the user.

To achieve our prototype requirements, it was necessary for us to acquire a regular sized bicycle rack, so that the testing scale could be more realistic. The prototype is operated by a sequence of steps, that are clearly stated on the display. The prototype also has one lock, this fulfills the requirement with the number of locks the prototype should handle. And finally, the prototype is compatible with existing RFID cards, we mainly used our student cards to test the system with.

The prototype is user-friendly and clearly states the directions for use on the display. The materials used for the prototype have proven to be sturdy and give realistic feedback to what the setup might be able to endure during a larger scale field test. After testing our prototype we can conclude that we have made functioning locking system, that allows a single user to lock their bicycle to the rack and associate it with their student ID card. Our prototype successfully demonstrates a single unit, where one user can access the device at a time. The larger scale implementation of this solution, is to use the same kind of display and card reader, but instead with greatly increased amount of locks along the bicycle rack. The built-in sensor would then recognize which lock has been locked, at intelligently associate it with a card.

The different risks from the risk management have all be taken into consideration. Some of the risk are too big for our prototype scope, but they will be taken into consideration in the further development of the product.

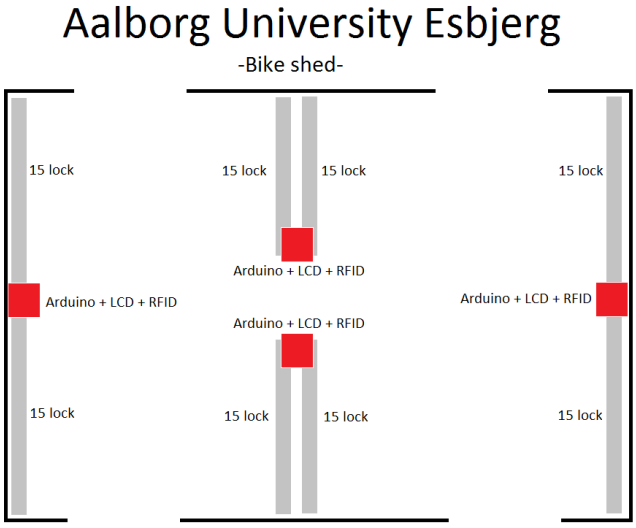


**Figure 5.1:** Aalborg University bike shed

The figure above shows how the single display and card reader would be associated with a greater number of locks.

One of the major flaws with our prototype is the actual security. The chain used to

lock the bicycle to rack is vulnerable to wire cutters. But since this is a prototype, we do not consider this security risk as a failure, but instead it needs to be addressed in future versions. It would be necessary to take a deeper lock at using different materials needed to make a secure chain lock and to re-write the software so that there are no risk of exploits. Since the RFID card reader reads every card type, it has the potential to steal peoples unencrypted data from their cards. This needs to be addressed in improved versions of the product, to avoid possible code manipulation in the product once it has been shipped.

We have successfully proven our hypothesis through our prototype. If the prototype is placed on the campus grounds, any given student at Aalborg University Esbjerg is able to lock their bicycle to the bicycle rack. This means that the bicycle in theory is not vulnerable to thieves carrying the bicycle away from its locked location. The prototype removes the inconvenience of carrying around a chain lock and encourages users to lock their bicycle to rack using the supplied setup.

# Chapter 6

# Conclusion

Based on the test results, we can conclude that our hypothesis stands correct, that picking up bicycles is harder when a proper bicycle-locking system is implemented. Our prototype validates the suggested solution, which is to implement an RFID-based locking mechanism on top of a bicycle rack. Throughout the analysis of the problem, we figured out that bicycle theft is a confirmed problem in Denmark due to the ineffective use of the existing bicycle racks. Using RFID technology for identification was a good choice, due to RFID is being widespread. Due to simplicity, of running the solenoid and the fact that we did not have to create any extra mechanical parts, easy accessibility, and running cost, we can conclude that the solenoid was a good choice for the locking mechanism, for the prototype and for future use.

Starting out by prototyping the individual components helped us to identify risks and potential issues. It helped us to identify the appropriate micro-controller, screen and libraries associated with them. We correctly limited the code to fit the scope of our prototype and to live up to the written requirements.

Building the prototype validates the suggested solution, which is to implement an RFID-based locking mechanism on top of a bicycle rack.

# Chapter 7

# Perspective

Our prototype shows a working idea. The final product must be designed with improved usability, durability, safety, and low environmental impact.

We focused on making a working prototype as our first goal, due to our limited tools, resources, and time. Hence, we were not concerned with durability, ergonomics and so forth.

We have to keep in mind, that a good idea and product does not sell by itself. We need to make a more in-depth analysis on possible revenue streams for the idea, on how it could be monetized in the future. This product would be sold to businesses, rather than to end-users, therefore the monetization needs to be fee-based or data-based. One potential revenue stream would be to gather anonymised data from the usage of the bike racks and peak hours, and sell it to governmental organizations, like the Danish Ministry of Transport. Another way would be to provide special customer support for the businesses we have a contract with and charge them a monthly- or yearly-fee. Partnerships with insurance companies could also be another potential source of income, where we would get a small percentage after each employee who is insured at the company or business to whom we sold the product, by implementing our safer bike locking solution.

All the suggestions above need to be investigated and researched more before getting to a conclusion on the revenue streams to our final product.

The steps of our further development would be:

1. A custom electronic board with a micro-controller best suited to this type of implementation. For the purpose of prototyping use of a development board like Arduino is convenient, the actual product needs a custom designed board, which saves costs of production.
2. Designing a robust locking enclosure. For the final product a design of a metal case is necessary to ensure durability and safety. The plywood lock is built as a proof of concept.
3. Extension of the system using extra measures such as video surveillance,

limited-access areas and other measures depending on location.

4. Integration of the system into RFID existing access and security systems. Our prototype is built as a independent unit.
5. The final product we aim to create, should be scalable, so that it could work with any number of locks.
6. Complying with existing directives and acts regarding electronic equipment.
7. Research and analysis of possible revenue streams.
8. Setting up a marketing plan and a business case for potential investors.
9. More thorough health and safety analysis of our product.

All these steps would most likely lead to a more usable, robust, and safe product. Still, it is important to understand issues, which would expose our product to the day-to-day environment.

Throughout the development of the prototype, we have not considered making a product that would replace every other bike-locking system that is currently in use. In fact, at time we considered creating a product which could coexist with the current systems, in this matter it is important to realize where we can implement our product. As mentioned in previous chapters, there are many places which have, and can make use of, existing RFID-equipped cards, which would make our system very convenient, and easy to use. These locations could be schools, universities, or train station's bike sheds. This seems to be promising, however it would require more responsibility regarding RFID cards' data scanned by system (fx, by data hashing, external database, and more) and bicycle safety.

With the growth of this product, it is clear that many concerns would arise in regards to the privacy issues of using RFID technology. Some people are very much against RFIDs because they perceive it as a breach into their personal lives. This may be due to misconceptions about how the RFID technology works, so we should keep this in mind, and perhaps try to educate people about exactly what they are using.

We have to keep in mind that our product will have societal effects, due to the broad possibilities of implementation. This could potentially change people's behaviour in locking the bicycles, using RFID cards and reducing thefts.

For future use, we have gone over some improvement in the code. It does not address all aspects of the final code, but goes over some of the bigger ones.

```
1  do {
2    successRead = getID(); // sets successRead to 1 when we get read
     from reader otherwise 0
3    sensorCheck = sensorRead(); //Read all sensors
4    if (programMode) {
5      programModeOn();        //program mode will be for us to put in
     new cards no use of led just LCD
6    }
7    else {
8      normalModeOn(); //Normal mode will just show the welcome screen
     on the LCD
9    }
10 }
11 while (!successRead || !sensorCheck); //the program will not go
     further while you not get a successful read
```

RFID V1 main loop

In the main loop of the final product it has a do while loop. It checks if a new RFID tag is read and if a new lock is put in. It has two displays and modes. A programming mode to put in new cards or change bugs. And then a normal mode. The normal-mode only shows the welcome screen. The welcome-screen is a scrolling display but only moves one letter at a time and then checks if a new lock or RFID tag has been checked in.

```
1 ////////////////////// Lock bike ////////////////////////////////
2 void lockLock( byte* RFID[], int sensor ){
3   //how it will lock needs code
4   *Lock = RFID = sensor
5
6   lockCheck[sensor-4]=1;
7
8 }
```

RFID V1 lockLock function

The function *lockLock*() associates a RFID tag to a lock. This function was not finished for this paper and would be a future work for the end product. The current system stores cards in the Arduino EEPROM, internal memory, but in future implementation the program would have a centralized database. Using a centralized database both helps with security since an Arduino can easily be stolen and the memory of the unit can be viewed. But with a database you can move it to a more secure location. Having a database will also add more features to the locking system, since a white-list could be put into use. The basic use of the *lockLock*() function would be to use a pointer byte array, $byteRFID[]$, to connect with a $storedCard[]$ array and connect those two to a lock number.

```
////////////////// Find ID From EEPROM   //////////////////////
boolean findLock( byte find[] ) {
  int count = EEPROM.read(0); // Read the first Byte of EEPROM that
  for ( int i = 1; i <= count; i++ ) {  // Loop once for each EEPROM
   entry

    readID(i); // Read an ID from EEPROM, it is stored in storedCard[4]

    if( checkTwo( find, storedCard ) ) {  // Check to see if the
   storedCard read from EEPROM
      return true;
      break; // Stop looking we found it
    }
    else {  // If not, return false
    }
  }
  return false;
}
```

RFID V1 findLock function

When a user would return and scan his card, the function $findLock()$ would then be looked up in the $lock[]$ array send it to the function $checkTwo()$. If the RFID is stored in the $lock[]$ array it would return true. The number of the lock would be found and display a message about unlocking the lock would appear to the user and it would be unlocked.

# Bibliography


[1] Krishnadev Calamur. In almost every european country, bikes are outselling new cars. http://www.npr.org/blogs/parallels/2013/10/24/240493422/in-most-every-european-country-bikes-are-outselling-cars, October 2014.

[2] John Pucher and Ralph Buehler. Making cycling irresistible: Lessons from the netherlands, denmark and germany. http://policy.rutgers.edu/faculty/pucher/Irresistible.pdf, July 2008.

[3] Statestikbanken. Straf11: Anmeldte forbrydelser efter område og overtrædelsesart. http://www.statistikbanken.dk/statbank5a/selectvarval/define.asp?PLanguage=0&subword=tabsel&MainTable=STRAF11&PXSId=146190&tablestyle=&ST=SD&buttons=0, September 2014.

[4] Søren Astrup. Her hugger cykeltyvene til: Ved stationer, skoler og biografer. http://politiken.dk/indland/ECE1652734/her-hugger-cykeltyvene-til-ved-stationer-skoler-og-biografer/, June 2012.

[5] Jeppe Lykke Hansen. Sådan stjæler de vores børns cykler. http://nyhederne.tv2.dk/article.php/id-58369224:video-s%C3%A5dan-stj%C3%A6ler-de-vores-b%C3%B8rns-cykler.html, October 2012.

[6] Picture of the most common lock type. https://upload.wikimedia.org/wikipedia/commons/2/21/Bike_O_Lock_Japan.jpg, November 2014.

[7] Mounted lock. http://image.made-in-china.com/2f0j00LCPtEaHFadqI/Bicycle-Lock.jpg, November 2014.

[8] Free mounted lock. http://www.wigglestatic.com/images/abus-umini-yellow-11-zoom.jpg, November 2014.

[9] Chain lock picture. http://www.justkryptonite.com/image/cache/data/hardwire_2510_key_cable_bicycle_lock-500x500.jpg, November 2014.

[10] Chain lock picture. http://www.photo-dictionary.com/photofiles/list/3616/5634bicycle_lock.jpg, November 2014.

[11] Rejsekort. http://www.rejsekort.dk/om-rejsekort/rejsekort-as.aspx, November 2014.

[12] Enterprise Surveys World Bank. Power outages in firms in a typical month. http://www.indexmundi.com/facts/indicators/IC.ELC.OUTG, April 2013.

[13] Council of the European Union European Parliament. Directive 2012/19/eu of the european parliament and of the council of 4 july 2012 on waste electrical and electronic equipment (weee) text with eea relevance. http://eur-lex.europa.eu/legal-content/EN/NOT/?uri=CELEX:32012L0019, July 2012.

[14] Heathland Ltd. Abs recycling. http://www.heathland.nl/abs-recycling.html, 2009.

[15] BRITISH STAINLESS STEEL ASSOCIATION. Environmental aspects of stainless steel.

[16] The Association for Science Education. Copper recycling and sustainability. http://resources.schoolscience.co.uk/CDA/16plus/sustainability/copper3.html.

[17] The Aluminium Association. Recycling. http://www.aluminum.org/industries/production/recycling.

[18] Council of the European Union European Parliament. Directive 2011/65/eu of the european parliament and of the council of 8 june 2011 on the restriction of the use of certain hazardous substances in electrical and electronic equipment text with eea relevance. http://eur-lex.europa.eu/legal-content/EN/NOT/?uri=CELEX:32011L0065, June 2011.

[19] Oberdörster et al. Nanotoxicology: An emerging discipline evolving from studies of ultrafine particles. http://www.scopus.com/record/display.url?eid=2-s2.0-20644449754&origin=inward&txGid=E9D00510448A78597696C74A529A29FB.CnvicAmOODVwpVrjSeqQ%3a1, July 2005.

[20] D. Halliday, R. Resnick, and J. Walker. Fundamentals of physics, 2010.

[21] Douglas Giancoli. Physics: principles with applications (5th ed.)., 1998.

[22] Danish Energy Agency. Energy statistics 2012. http://www.ens.dk/sites/ens.dk/files/info/tal-kort/statistik-noegletal/aarlig-energistatistik/energy_statistics_2012.pdf, February 2014.

[23] Inc. Hitec RCD USA. Rf-020th. http://www.mabuchi-motor.co.jp/en_US/cat_files/rf_020th.pdf.

[24] Danish Energy Agency. Hs-805bb. http://www.robotshop.com/media/files/pdf/hs805.pdf.

[25] Adafruit. Lock-style solenoid - 12vdc. http://www.adafruit.com/products/1512#technical-details-anchor.

[26] Daniel M. Dobkin. *The RF in RFID*. Newnes, 2012.

[27] Makecourse. Rfid week 10. https://www.youtube.com/watch?v=hxSQmTkIGAs, February 2012.

[28] Wikimedia. Relay symbols. https://upload.wikimedia.org/wikipedia/commons/6/67/Relay_symbols.svg, November 2014.

[29] Electronics Tutorials. The electrical relay. http://www.electronics-tutorials.ws/io/io_5.html, January 2014.

[30] Rfid reader datasheet. http://www.nxp.com/documents/data_sheet/MFRC522.pdf.

[31] Solenoid datasheet. http://www.adafruit.com/products/1512#tab_technical-details.

[32] Microcontroller datasheet. http://arduino.cc/en/Main/arduinoBoardUno.

[33] Switch datasheet. http://www.qiaoh.com/en/product-show.asp?id=128.

[34] Lcd datasheet. http://www.farnell.com/datasheets/1536558.pdf.

# Chapter 8

# Appendix

## 8.1 Lock data

Raw data from lock type data gathering. The data gathering took place at Aalborg University Esbjerg's bicycle sheds on Tuesday November 11th from 9.00AM to 10.00AM.

| | |
|---|---|
| No lock | 1 |
| Only Mounted lock | 103 |
| Only Chain lock | 35 |
| Both lock types | 15 |
| Locked to Rack | 8 |
| **Total bicycles** | **154** |

## 8.2 Codes and header files

### 8.2.1 Prototype code

```
////////// AAUE bike locking systen /////////////////
/*
This program is based on two older projects LCD and RFID door locking system.
#1 LCD - https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home
#2 RFID - https://github.com/miguelbalboa/rfid

It is a P1 project in Aalborg University Esbjerg, in Electrical and Computer Engingeering. It uses
a RFID reader to lock and unlock a bike rack. It was made to solve the problem of people stealing
locked bikes that were not locked to a bike rack. The thiefs would take the bikes and throw them
into cars and drive away. In private they would break up the locks on the bikes.

*/

////////// Arduino Basic libraries ////////////
//These libraries are basic use with the arduino and can all be looked up at the arduino.cc webpage
#include <Wire.h>
#include <LCD.h>
#include <EEPROM.h>  // We are going to read and write PICC's UIDs from/to EEPROM
#include <SPI.h>      // RC522 Module uses SPI protocol
//////// Extra imported libraries ////////////
//These are extra libraries that we got to make the system work. We will include them in appendix
#include <MFRC522.h>   // Library for Mifare RC522 Devices
#include <LiquidCrystal_I2C.h> // library for I2C LCD display

#define I2C_ADDR    0x27  // Define I2C Address where the PCF8574A is
#define BACKLIGHT_PIN     3
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
#define D6_pin  6
#define D7_pin  7
#define sensor 3
#define relay 4
#define wipeB 8 // Button pin for WipeMode
#define SS_PIN 10
#define RST_PIN 9

boolean match = false; // initialize card match to false
boolean programMode = false; // initialize programming mode to false

int successRead; // Variable integer to keep if we have Successful Read from Reader
int lcdDelay1, lcdDelay2;
int notLocked;
int lcdScroll;
int scrollCounter;
int scrollLeft;
int val;
int lcdLockPrint;
int sensorCheck;

byte storedCard[4];   // Stores an ID read from EEPROM
byte readCard[4];           // Stores scanned ID read from RFID Module
byte masterCard[4]; // Stores master card's ID read from EEPROM
byte* lock[4];

int lockCheck[5];

/* We need to define MFRC522's pins and create instance
 * Pin layout should be as follows (on Arduino Uno):
 * MOSI: Pin 11 / ICSP-4
 * MISO: Pin 12 / ICSP-1
 * SCK : Pin 13 / ICSP-3
 * SS : Pin 10 (Configurable)
 * RST : Pin 9 (Configurable)
 * look MFRC522 Library for
 * pin configuration for other Arduinos.
 */

LiquidCrystal_I2C lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance.


void setup() {

  lcd.begin (16,2);
  ///////////////// LCD setup /////////////////
  // Switch on the backlight
  lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
  lcd.setBacklight(HIGH);
```

```
83
   ////////////////// RFID setup ///////////////
85
   //Arduino Pin Configuration
87
   pinMode(sensor, INPUT);
89
   for (int i=4; i<=relay; i++) {
91   pinMode(i, OUTPUT);
     digitalWrite(i, LOW); // Make sure door is locked
93 }
95 //Protocol Configuration
   Serial.begin(9600);  // Initialize serial communications with PC
97 SPI.begin();           // MFRC522 Hardware uses SPI protocol
   mfrc522.PCD_Init();    // Initialize MFRC522 Hardware
99
   ////////////// Wibe mode setup //////////////////////
101
   //Wipe Code if Button Pressed while setup run (powered on) it wipes EEPROM
103 pinMode(wipeB, INPUT_PULLUP);  // Enable pin's pull up resistor
   if (digitalRead(wipeB) == LOW) {     // when button pressed pin should get low, button
      connected to ground
105   lcd.clear();
     lcd.home();                    // go home
107   lcd.print("Wibe mode on");
     lcd.setCursor ( 0, 1 );         // go to the 2nd line
109   lcd.print("in 5 second");
     delay(5000);     // Give user enough time to cancel operation
111
     if (digitalRead(wipeB) == LOW) {  // If button still be pressed, wipe EEPROM
113    lcd.clear();
       lcd.home();                     // go home
115    lcd.print("Wive mode");
       lcd.setCursor ( 0, 1 );         // go to the 2nd line
117    lcd.print("Started now");
       for (int x=0; x<1024; x=x+1){ //Loop end of EEPROM address
119      if (EEPROM.read(x) == 0){ //If EEPROM address 0
           // do nothing, already clear, go to the next address in order to save time and reduce
     writes to EEPROM
121      }
         else{
123        EEPROM.write(x, 0); // if not write 0, it takes 3.3mS
         }
125    }
       lcd.clear();
127    lcd.home();                     // go home
       lcd.print("Wibe mode");
129    lcd.setCursor ( 0, 1 );         // go to the 2nd line
       lcd.print("EEPROM wibed");
131    delay(2000);
     }
133  else {
       lcd.clear();
135    lcd.home();                     // go home
       lcd.print("Wibe mode");
137    lcd.setCursor ( 0, 1 );         // go to the 2nd line
       lcd.print("Cancelled");
139    delay(2000);
     }
141 }
143 //Check if master card defined, if not let user choose a master card
   //This also useful to just redefine Master Card
145 //You can keep other EEPROM records just write other than 1 to EEPROM address 1
   if (EEPROM.read(1) != 1) {  // Look EEPROM if Master Card defined, EEPROM address 1 holds if
      defined
147   lcd.clear();
     lcd.home();                    // go home
149   lcd.print("No master card");
     lcd.setCursor ( 0, 1 );         // go to the 2nd line
151   lcd.print("Scan new card");

153   do {
       successRead = getID(); // sets successRead to 1 when we get read from reader otherwise 0
155  }
     while (!successRead); //the program will not go further while you not get a successful read
157  for ( int j = 0; j < 4; j++ ) { // Loop 4 times
       EEPROM.write( 2 +j, readCard[j] ); // Write scanned PICC's UID to EEPROM, start from
      address 3
159  }
     EEPROM.write(1,1); //Write to EEPROM we defined Master Card.
161  lcd.clear();
     lcd.home();                    // go home
163  lcd.print("New master card");
     lcd.setCursor ( 0, 1 );         // go to the 2nd line
165  lcd.print("Has been entered");
     delay(2000);
167 }
   for ( int i = 0; i < 4; i++ ) { // Read Master Card's UID from EEPROM
169  masterCard[i] = EEPROM.read(2+i); //Write it to masterCard
     Serial.print(masterCard[i]);
171 }
}
173
```

```
////////////////////////////// Main Loop ////////////////////////////////
175
void loop () {
177     successRead = getID(); // sets successRead to 1 when we get read from reader otherwise 0
        sensorCheck = sensorRead(); //Read all sensors if new sensor read then enter locking mode
179     welcomeScreen();

181   if ( successRead ) {
        if ( findID(readCard) ) {        // If not, see if the card is in the EEPROM
183       openLock(500, 0);
          deleteID(readCard);
185       // Open the door lock for 5000 ms
        }
187     else {        // If not, show that the ID was not valid
          failed();
189     }

191   }

193   if ( sensorCheck ) {
        askRFID();
195     for (int i = 0; i <= 600; i++) {
          if (i < 600){
197         successRead = getID();
            if (successRead) {
199           notLocked=0;
              break;
201         }
          }
203       else {
            failedLock();
205         openLock(500, sensorCheck);
          }
207       delay(50);
        }
209
        if (notLocked == 0){
211       writeID(readCard);
          lockLock(sensorCheck);
213       delay(3000);

215       lcdDelay2=0;
        }
217     else {
          lcdDelay2=0;
219     }
    }
221 }

223 ////////////////////////// Get PICC's UID ///////////////////////

225 int sensorRead(){
        //This goes over all the sensors to see a change in HIGH and LOW
227     if (lockCheck[0] == 0) { //if a sensor has a high signal but has been connectd to a RFID tag
          then it's ignored
          val = digitalRead(3);
229       if (val == 1) return 1;
          else return 0;
231     }
        return 0;
233 }

235 ////////////////////////// Get PICC's UID ///////////////////////

237 int getID() {
      // Getting ready for Reading PICCs
239   if ( ! mfrc522.PICC_IsNewCardPresent()) { //If a new PICC placed to RFID reader continue
        return 0;
241   }
      if ( ! mfrc522.PICC_ReadCardSerial()) { //Since a PICC placed get Serial and continue
243     return 0;
      }
245   // There are Mifare PICCs which have 4 byte or 7 byte UID care if you use 7 byte PICC;
      for (int i = 0; i < mfrc522.uid.size; i++) {  // for size of uid.size write uid.uidByte to
         readCard
247     readCard[i] = mfrc522.uid.uidByte[i];
      }
249   mfrc522.PICC_HaltA(); // Stop reading
      return 1;
251 }

253 ////////////////////// Normal Mode Leds  ////////////////////////

255 void welcomeScreen () {
      //This is the welcome screen that will be scrolling while no RFID or new locked is used.
257   if (lcdDelay2 == 0){

259   lcd.clear();
      lcd.home();                    // go home
261   lcd.print("Aalborg University Esbjerg");
      lcd.setCursor ( 0, 1 );         // go to the 2nd line
263   lcd.print(" RFID Bike Locking System");

265   lcdScroll=1;
      scrollCounter=0;
```

```
267   lcdDelay2=1;
      scrollLeft=0;
269
      }
271
      if (lcdScroll == 1) {//We want to read a new RFID tag or check sensor every 50 ms so we only
         move one letter at a time
273
        if (scrollCounter >= 5 && scrollCounter <= 14) {
275
          if (scrollLeft == 4){
277         // scroll one position left:
            lcd.scrollDisplayLeft();
279         scrollCounter++;
            scrollLeft=0;
281       }

283       else {
            scrollLeft++;
285       }
        }
287
        else if (scrollCounter <= 4 && scrollCounter >= 0 ) {
289
          if (scrollLeft == 4){
291         scrollCounter++;
            scrollLeft=0;
293       }

295       else {
            scrollLeft++;
297       }
        }
299
        else{
301       lcdDelay2=0;
          lcdScroll=0;
303       delay(200);
        }
305     delay (50);
      }
307 }

309 ////////////////////// Read an ID from EEPROM //////////////////////

311 void readID( int number ) {
      int start = (number * 4 ) + 2; // Figure out starting position
313   for ( int i = 0; i < 4; i++ ) { // Loop 4 times to get the 4 Bytes
        storedCard[i] = EEPROM.read(start+i); // Assign values read from EEPROM to array
315   }
    }
317
    ///////////////// Add ID to EEPROM   ///////////////////////////////

319
    void writeID( byte a[] ) {
321   if ( !findID( a ) ) { // Before we write to the EEPROM, check to see if we have seen this card
         before!
        int num = EEPROM.read(0); // Get the numer of used spaces, position 0 stores the number of
        ID cards
323     int start = ( num * 4 ) + 6; // Figure out where the next slot starts
        num++; // Increment the counter by one
325     EEPROM.write( 0, num ); // Write the new count to the counter
        for ( int j = 0; j < 4; j++ ) { // Loop 4 times
327       EEPROM.write( start+j, a[j] ); // Write the array values to EEPROM in the right position
        }
329   }
    }
331
    ////////////////////// Remove ID from EEPROM   //////////////////////////

333
    void deleteID( byte a[] ) {
335   if ( !findID( a ) ) { // Before we delete from the EEPROM, check to see if we have this card!
      }
337   else {
        int num = EEPROM.read(0); // Get the numer of used spaces, position 0 stores the number of
        ID cards
339     int slot; // Figure out the slot number of the card
        int start;// = ( num * 4 ) + 6; // Figure out where the next slot starts
341     int looping; // The number of times the loop repeats
        int j;
343     int count = EEPROM.read(0); // Read the first Byte of EEPROM that stores number of cards
        slot = findIDSLOT( a ); //Figure out the slot number of the card to delete
345     start = (slot * 4) + 2;
        looping = ((num - slot) * 4);
347     num--; // Decrement the counter by one
        EEPROM.write( 0, num ); // Write the new count to the counter
349     for ( j = 0; j < looping; j++ ) { // Loop the card shift times
          EEPROM.write( start+j, EEPROM.read(start+4+j)); // Shift the array values to 4 places
        earlier in the EEPROM
351     }
        for ( int k = 0; k < 4; k++ ) { //Shifting loop
353       EEPROM.write( start+j+k, 0);
        }
355   }
    }
```

```
357
////////////////////// Check Bytes   ///////////////////////////
359
boolean checkTwo ( byte a[], byte b[] ) {
361   if ( a[0] != NULL ) // Make sure there is something in the array first
        match = true; // Assume they match at first
363   for ( int k = 0; k < 4; k++ ) { // Loop 4 times
        if ( a[k] != b[k] ) // IF a != b then set match = false, one fails, all fail
365       match = false;
      }
367   if ( match ) { // Check to see if if match is still true
        return true; // Return true
369   }
      else  {
371     return false; // Return false
      }
373 }

375 ////////////////////// Find Slot   //////////////////////////

377 int findIDSLOT( byte find[] ) {
      int count = EEPROM.read(0); // Read the first Byte of EEPROM that
379   for ( int i = 1; i <= count; i++ ) { // Loop once for each EEPROM entry
        readID(i); // Read an ID from EEPROM, it is stored in storedCard[4]
381     if( checkTwo( find, storedCard ) ) { // Check to see if the storedCard read from EEPROM
          // is the same as the find[] ID card passed
383       return i; // The slot number of the card
          break; // Stop looking we found it
385     }
      }
387 }

389 ////////////////////////////////// Find ID From EEPROM ////////////////////

391 boolean findID( byte find[] ) {
      int count = EEPROM.read(0); // Read the first Byte of EEPROM that
393   for ( int i = 1; i <= count; i++ ) {  // Loop once for each EEPROM entry
        readID(i); // Read an ID from EEPROM, it is stored in storedCard[4]
395     if( checkTwo( find, storedCard ) ) {  // Check to see if the storedCard read from EEPROM
          return true;
397       break; // Stop looking we found it
        }
399     else {  // If not, return false
        }
401   }
      return false;
403 }

405 /////////////// Check readCard IF is masterCard   ///////////////////

407 boolean isMaster( byte test[] ) { // Check to see if the ID passed is the master programing card
      if ( checkTwo( test, masterCard ) )
409     return true;
      else
411     return false;
413 }

415 ////////////////////// Lock bike ////////////////////////////////

417 void lockLock( int locksensor ){
      //Message to user if he has used a RFID to lock a new lock
      lcd.clear();
419   lcd.home();                    // go home)
      lcd.print("1 Is now locked");
421   lcd.setCursor ( 0, 1 );        // go to the 2nd line
      lcd.print("   Thank you");
423
      lcdDelay2 = 0;
425
      lockCheck[0]=1;
427
429 }

431 ////////////////////////// Unlock bike   ////////////////////

433 void openLock( int setDelay, int locksensor ) {
      //two messages if a user failes to use RFID tag or opens his lock
      if (lcdLockPrint == 1) {
435     digitalWrite(relay, HIGH); // Unlock bike
        delay(setDelay); // Hold lock open for given seconds
437     digitalWrite(relay, LOW);
        lcdDelay2 = 0;
439     lcdLockPrint = 0;
      }
441   else {
        digitalWrite(relay, HIGH); // Unlock bike
443     lcd.clear();
        lcd.home();                    // go home
445     lcd.print("1 Is now open");
        lcd.setCursor ( 0, 1 );        // go to the 2nd line
447     lcd.print("Have a nice day");
        delay(setDelay); // Hold door lock open for given seconds
449     digitalWrite(relay, LOW);
        lcdDelay2 = 0;
451   }
```

```
453   lockCheck[0]=0;

455 }

457 ////////////////////  Failed Locking  ////////////////////

459 void failedLock() {
      //Message to user that locking failed no RFID tag
461   lcd.clear();
      lcd.home();                     // go home
463   lcd.print("Unlocked lock 1");
      lcd.setCursor ( 0, 1 );         // go to the 2nd line
465   lcd.print("No RFID tag used");

467   notLocked=1;
      lcdLockPrint = 1;
469   lcdDelay2=0;
    }
471
    ///////////////  Failed Access  ////////////////////////
473
    void failed() {
475   //Message to user that system cant read his RFID tag or it has been used before
      lcd.clear();
477   lcd.home();                     // go home
      lcd.print("Error, RFID not");
479   lcd.setCursor ( 0, 1 );         // go to the 2nd line
      lcd.print("known or used");
481   delay(1000);

483   lcdDelay2=0;
    }
485
    /////////////////  LC read RFID ///////////////////////
487
    void askRFID() {
489   //Message to use to confirm that it is the correct lock and he needs to scan his card.
      lcd.clear();
491   lcd.home();                     // go home
      lcd.print("Is lock 1 yours");
493   lcd.setCursor ( 0, 1 );         // go to the 2nd line
      lcd.print("Then scan IDcard");
495 }
```

Prototype full code

### 8.2.2 MRFC522 RFID reader header file for main code

```
/**
 * MFRC522.h - Library to use ARDUINO RFID MODULE KIT 13.56 MHZ WITH TAGS SPI W AND R BY COOQROBOT.
 * Based on code Dr.Leong   ( WWW.B2CQSHOP.COM )
 * Created by Miguel Balboa (circuitito.com), Jan, 2012.
 * Rewritten by Soeren Thing Andersen (access.thing.dk), fall of 2013 (Translation to English, refactored, comments, anti collision, cascade levels.)
 * Released into the public domain.
 *
 * Please read this file for an overview and then MFRC522.cpp for comments on the specific functions.
 * Search for "mf-rc522" on ebay.com to purchase the MF-RC522 board.
 *
 * There are three hardware components involved:
 * 1) The micro controller: An Arduino
 * 2) The PCD (short for Proximity Coupling Device): NXP MFRC522 Contactless Reader IC
 * 3) The PICC (short for Proximity Integrated Circuit Card): A card or tag using the ISO 14443A interface, eg Mifare or NTAG203.
 *
 * The microcontroller and card reader uses SPI for communication.
 * The protocol is described in the MFRC522 datasheet: http://www.nxp.com/documents/data_sheet/MFRC522.pdf
 *
 * The card reader and the tags communicate using a 13.56MHz electromagnetic field.
 * The protocol is defined in ISO/IEC 14443-3 Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 3: Initialization and anticollision".
 * A free version of the final draft can be found at http://wg8.de/wg8n1496_17n3613_Ballot_FCD14443-3.pdf
 * Details are found in chapter 6, Typa A - Initialization and anti-collision.
 *
 * If only the PICC UID is wanted, the above documents has all the needed information.
 * To read and write from MIFARE PICCs, the MIFARE protocol is used after the PICC has been selected.
 * The MIFARE Classic chips and protocol is described in the datasheets:
 *     1K:   http://www.nxp.com/documents/data_sheet/MF1S503x.pdf
 *     4K:   http://www.nxp.com/documents/data_sheet/MF1S703x.pdf
 *     Mini: http://www.idcardmarket.com/download/mifare_S20_datasheet.pdf
 * The MIFARE Ultralight chip and protocol is described in the datasheets:
 *     Ultralight:   http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf
 *     Ultralight C: http://www.nxp.com/documents/short_data_sheet/MF0ICU2_SDS.pdf
 *
 * MIFARE Classic 1K (MF1S503x):
 *     Has 16 sectors * 4 blocks/sector * 16 bytes/block = 1024 bytes.
 *     The blocks are numbered 0-63.
 *     Block 3 in each sector is the Sector Trailer. See http://www.nxp.com/documents/data_sheet/MF1S503x.pdf sections 8.6 and 8.7:
 *           Bytes 0-5:   Key A
 *           Bytes 6-8:   Access Bits
 *           Bytes 9:     User data
 *           Bytes 10-15: Key B (or user data)
 *     Block 0 is read only manufacturer data.
 *     To access a block, an authentication using a key from the block's sector must be performed first.
 *     Example: To read from block 10, first authenticate using a key from sector 3 (blocks 8-11).
 *     All keys are set to FFFFFFFFFFFFh at chip delivery.
 *     Warning: Please read section 8.7 "Memory Access". It includes this text: if the PICC detects a format violation the whole sector is irreversibly blocked.
 *     To use a block in "value block" mode (for Increment/Decrement operations) you need to change the sector trailer. Use PICC_SetAccessBits() to calculate the bit patterns.
 * MIFARE Classic 4K (MF1S703x):
 *     Has (32 sectors * 4 blocks/sector + 8 sectors * 16 blocks/sector) * 16 bytes/block = 4096 bytes.
 *     The blocks are numbered 0-255.
 *     The last block in each sector is the Sector Trailer like above.
 * MIFARE Classic Mini (MF1 IC S20):
 *     Has 5 sectors * 4 blocks/sector * 16 bytes/block = 320 bytes.
 *     The blocks are numbered 0-19.
 *     The last block in each sector is the Sector Trailer like above.
 *
 * MIFARE Ultralight (MF0ICU1):
 *     Has 16 pages of 4 bytes = 64 bytes.
 *     Pages 0 + 1 is used for the 7-byte UID.
 *     Page 2 contains the last chech digit for the UID, one byte manufacturer internal data, and the lock bytes (see http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf section 8.5.2)
 *     Page 3 is OTP, One Time Programmable bits. Once set to 1 they cannot revert to 0.
 *     Pages 4-15 are read/write unless blocked by the lock bytes in page 2.
 * MIFARE Ultralight C (MF0ICU2):
 *     Has 48 pages of 4 bytes = 64 bytes.
 *     Pages 0 + 1 is used for the 7-byte UID.
 *     Page 2 contains the last chech digit for the UID, one byte manufacturer internal data, and the lock bytes (see http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf section 8.5.2)
 *     Page 3 is OTP, One Time Programmable bits. Once set to 1 they cannot revert to 0.
 *     Pages 4-39 are read/write unless blocked by the lock bytes in page 2.
 *     Page 40 Lock bytes
 *     Page 41 16 bit one way counter
 *     Pages 42-43 Authentication configuration
 *     Pages 44-47 Authentication key
 */
#ifndef MFRC522_h
#define MFRC522_h
```

```
77 #include <Arduino.h>
   #include <SPI.h>
79
   class MFRC522 {
81 public:
     // MFRC522 registers. Described in chapter 9 of the datasheet.
83   // When using SPI all addresses are shifted one bit left in the "SPI address byte" (section
       8.1.2.3)
     enum PCD_Register {
85     // Page 0: Command and status
       //              0x00      // reserved for future use
87     CommandReg        = 0x01 << 1,  // starts and stops command execution
       ComIEnReg       = 0x02 << 1,  // enable and disable interrupt request control bits
89     DivIEnReg       = 0x03 << 1,  // enable and disable interrupt request control bits
       ComIrqReg       = 0x04 << 1,  // interrupt request bits
91     DivIrqReg       = 0x05 << 1,  // interrupt request bits
       ErrorReg        = 0x06 << 1,  // error bits showing the error status of the last command
        executed
93     Status1Reg        = 0x07 << 1,  // communication status bits
       Status2Reg        = 0x08 << 1,  // receiver and transmitter status bits
95     FIFODataReg       = 0x09 << 1,  // input and output of 64 byte FIFO buffer
       FIFOLevelReg      = 0x0A << 1,  // number of bytes stored in the FIFO buffer
97     WaterLevelReg     = 0x0B << 1,  // level for FIFO underflow and overflow warning
       ControlReg        = 0x0C << 1,  // miscellaneous control registers
99     BitFramingReg     = 0x0D << 1,  // adjustments for bit-oriented frames
       CollReg         = 0x0E << 1,  // bit position of the first bit-collision detected on the RF
        interface
101    //              0x0F      // reserved for future use

103    // Page 1: Command
       //              0x10      // reserved for future use
105    ModeReg         = 0x11 << 1,  // defines general modes for transmitting and receiving
       TxModeReg       = 0x12 << 1,  // defines transmission data rate and framing
107    RxModeReg       = 0x13 << 1,  // defines reception data rate and framing
       TxControlReg      = 0x14 << 1,  // controls the logical behavior of the antenna driver pins
        TX1 and TX2
109    TxASKReg        = 0x15 << 1,  // controls the setting of the transmission modulation
       TxSelReg        = 0x16 << 1,  // selects the internal sources for the antenna driver
111    RxSelReg        = 0x17 << 1,  // selects internal receiver settings
       RxThresholdReg      = 0x18 << 1,  // selects thresholds for the bit decoder
113    DemodReg        = 0x19 << 1,  // defines demodulator settings
       //              0x1A      // reserved for future use
115    //              0x1B      // reserved for future use
       MfTxReg         = 0x1C << 1,  // controls some MIFARE communication transmit parameters
117    MfRxReg         = 0x1D << 1,  // controls some MIFARE communication receive parameters
       //              0x1E      // reserved for future use
119    SerialSpeedReg      = 0x1F << 1,  // selects the speed of the serial UART interface

121    // Page 2: Configuration
       //              0x20      // reserved for future use
123    CRCResultRegH     = 0x21 << 1,  // shows the MSB and LSB values of the CRC calculation
       CRCResultRegL     = 0x22 << 1,
125    //              0x23      // reserved for future use
       ModWidthReg       = 0x24 << 1,  // controls the ModWidth setting?
127    //              0x25      // reserved for future use
       RFCfgReg        = 0x26 << 1,  // configures the receiver gain
129    GsNReg          = 0x27 << 1,  // selects the conductance of the antenna driver pins TX1 and
        TX2 for modulation
       CWGsPReg        = 0x28 << 1,  // defines the conductance of the p-driver output during
        periods of no modulation
131    ModGsPReg       = 0x29 << 1,  // defines the conductance of the p-driver output during
        periods of modulation
       TModeReg        = 0x2A << 1,  // defines settings for the internal timer
133    TPrescalerReg     = 0x2B << 1,  // the lower 8 bits of the TPrescaler value. The 4 high bits
         are in TModeReg.
       TReloadRegH       = 0x2C << 1,  // defines the 16-bit timer reload value
135    TReloadRegL       = 0x2D << 1,
       TCounterValueRegH   = 0x2E << 1,  // shows the 16-bit timer value
137    TCounterValueRegL   = 0x2F << 1,

139    // Page 3: Test Registers
       //              0x30      // reserved for future use
141    TestSel1Reg       = 0x31 << 1,  // general test signal configuration
       TestSel2Reg       = 0x32 << 1,  // general test signal configuration
143    TestPinEnReg      = 0x33 << 1,  // enables pin output driver on pins D1 to D7
       TestPinValueReg     = 0x34 << 1,  // defines the values for D1 to D7 when it is used as an I
        /O bus
145    TestBusReg        = 0x35 << 1,  // shows the status of the internal test bus
       AutoTestReg       = 0x36 << 1,  // controls the digital self test
147    VersionReg        = 0x37 << 1,  // shows the software version
       AnalogTestReg     = 0x38 << 1,  // controls the pins AUX1 and AUX2
149    TestDAC1Reg       = 0x39 << 1,  // defines the test value for TestDAC1
       TestDAC2Reg       = 0x3A << 1,  // defines the test value for TestDAC2
151    TestADCReg        = 0x3B << 1   // shows the value of ADC I and Q channels
       //              0x3C      // reserved for production tests
153    //              0x3D      // reserved for production tests
       //              0x3E      // reserved for production tests
155    //              0x3F      // reserved for production tests
   };
157
   // MFRC522 commands. Described in chapter 10 of the datasheet.
159 enum PCD_Command {
     PCD_Idle          = 0x00,   // no action, cancels current command execution
161  PCD_Mem           = 0x01,   // stores 25 bytes into the internal buffer
     PCD_GenerateRandomID  = 0x02,   // generates a 10-byte random ID number
```

```
163     PCD_CalcCRC         = 0x03,   // activates the CRC coprocessor or performs a self test
        PCD_Transmit        = 0x04,   // transmits data from the FIFO buffer
165     PCD_NoCmdChange     = 0x07,   // no command change, can be used to modify the CommandReg
         register bits without affecting the command, for example, the PowerDown bit
        PCD_Receive         = 0x08,   // activates the receiver circuits
167     PCD_Transceive      = 0x0C,   // transmits data from FIFO buffer to antenna and
         automatically activates the receiver after transmission
        PCD_MFAuthent       = 0x0E,   // performs the MIFARE standard authentication as a reader
169     PCD_SoftReset       = 0x0F    // resets the MFRC522
    };
171
    // MFRC522 RxGain[2:0] masks, defines the receiver's signal voltage gain factor (on the PCD).
173 // Described in 9.3.3.6 / table 98 of the datasheet at http://www.nxp.com/documents/data_sheet
       /MFRC522.pdf
    enum PCD_RxGain {
175     RxGain_18dB         = 0x00 << 4,  // 000b - 18 dB, minimum
        RxGain_23dB         = 0x01 << 4,  // 001b - 23 dB
177     RxGain_18dB_2       = 0x02 << 4,  // 010b - 18 dB, it seems 010b is a duplicate for 000b
        RxGain_23dB_2       = 0x03 << 4,  // 011b - 23 dB, it seems 011b is a duplicate for 001b
179     RxGain_33dB         = 0x04 << 4,  // 100b - 33 dB, average, and typical default
        RxGain_38dB         = 0x05 << 4,  // 101b - 38 dB
181     RxGain_43dB         = 0x06 << 4,  // 110b - 43 dB
        RxGain_48dB         = 0x07 << 4,  // 111b - 48 dB, maximum
183     RxGain_min          = 0x00 << 4,  // 000b - 18 dB, minimum, convenience for RxGain_18dB
        RxGain_avg          = 0x04 << 4,  // 100b - 33 dB, average, convenience for RxGain_33dB
185     RxGain_max          = 0x07 << 4   // 111b - 48 dB, maximum, convenience for RxGain_48dB
    };
187
    // Commands sent to the PICC.
189 enum PICC_Command {
      // The commands used by the PCD to manage communication with several PICCs (ISO 14443-3,
       Type A, section 6.4)
191   PICC_CMD_REQA       = 0x26,   // REQuest command, Type A. Invites PICCs in state IDLE to go to
        READY and prepare for anticollision or selection. 7 bit frame.
      PICC_CMD_WUPA       = 0x52,   // Wake-UP command, Type A. Invites PICCs in state IDLE and HALT
        to go to READY(*) and prepare for anticollision or selection. 7 bit frame.
193   PICC_CMD_CT         = 0x88,   // Cascade Tag. Not really a command, but used during anti
       collision.
      PICC_CMD_SEL_CL1    = 0x93,   // Anti collision/Select, Cascade Level 1
195   PICC_CMD_SEL_CL2    = 0x95,   // Anti collision/Select, Cascade Level 1
      PICC_CMD_SEL_CL3    = 0x97,   // Anti collision/Select, Cascade Level 1
197   PICC_CMD_HLTA       = 0x50,   // HaLT command, Type A. Instructs an ACTIVE PICC to go to state
        HALT.
      // The commands used for MIFARE Classic (from http://www.nxp.com/documents/data_sheet/
       MF1S503x.pdf, Section 9)
199   // Use PCD_MFAuthent to authenticate access to a sector, then use these commands to read/
       write/modify the blocks on the sector.
      // The read/write commands can also be used for MIFARE Ultralight.
201   PICC_CMD_MF_AUTH_KEY_A  = 0x60,   // Perform authentication with Key A
      PICC_CMD_MF_AUTH_KEY_B  = 0x61,   // Perform authentication with Key B
203   PICC_CMD_MF_READ    = 0x30,   // Reads one 16 byte block from the authenticated sector of
       the PICC. Also used for MIFARE Ultralight.
      PICC_CMD_MF_WRITE   = 0xA0,   // Writes one 16 byte block to the authenticated sector of the
        PICC. Called "COMPATIBILITY WRITE" for MIFARE Ultralight.
205   PICC_CMD_MF_DECREMENT = 0xC0,   // Decrements the contents of a block and stores the result
       in the internal data register.
      PICC_CMD_MF_INCREMENT = 0xC1,   // Increments the contents of a block and stores the result
       in the internal data register.
207   PICC_CMD_MF_RESTORE   = 0xC2,   // Reads the contents of a block into the internal data
       register.
      PICC_CMD_MF_TRANSFER  = 0xB0,   // Writes the contents of the internal data register to a
       block.
209   // The commands used for MIFARE Ultralight (from http://www.nxp.com/documents/data_sheet/
       MF0ICU1.pdf, Section 8.6)
      // The PICC_CMD_MF_READ and PICC_CMD_MF_WRITE can also be used for MIFARE Ultralight.
211   PICC_CMD_UL_WRITE   = 0xA2    // Writes one 4 byte page to the PICC.
    };
213
    // MIFARE constants that does not fit anywhere else
215 enum MIFARE_Misc {
      MF_ACK            = 0xA,    // The MIFARE Classic uses a 4 bit ACK/NAK. Any other value than 0
       xA is NAK.
217   MF_KEY_SIZE         = 6       // A Mifare Crypto1 key is 6 bytes.
    };
219
    // PICC types we can detect. Remember to update PICC_GetTypeName() if you add more.
221 enum PICC_Type {
      PICC_TYPE_UNKNOWN     = 0,
223   PICC_TYPE_ISO_14443_4 = 1,  // PICC compliant with ISO/IEC 14443-4
      PICC_TYPE_ISO_18092   = 2,  // PICC compliant with ISO/IEC 18092 (NFC)
225   PICC_TYPE_MIFARE_MINI = 3,  // MIFARE Classic protocol, 320 bytes
      PICC_TYPE_MIFARE_1K   = 4,  // MIFARE Classic protocol, 1KB
227   PICC_TYPE_MIFARE_4K   = 5,  // MIFARE Classic protocol, 4KB
      PICC_TYPE_MIFARE_UL   = 6,  // MIFARE Ultralight or Ultralight C
229   PICC_TYPE_MIFARE_PLUS = 7,  // MIFARE Plus
      PICC_TYPE_TNP3XXX     = 8,  // Only mentioned in NXP AN 10833 MIFARE Type Identification
       Procedure
231   PICC_TYPE_NOT_COMPLETE  = 255 // SAK indicates UID is not complete.
    };
233
    // Return codes from the functions in this class. Remember to update GetStatusCodeName() if
       you add more.
235 enum StatusCode {
      STATUS_OK         = 1,  // Success
237   STATUS_ERROR      = 2,  // Error in communication
```

```
    STATUS_COLLISION    = 3,  // Collission detected
    STATUS_TIMEOUT      = 4,  // Timeout in communication.
    STATUS_NO_ROOM      = 5,  // A buffer is not big enough.
    STATUS_INTERNAL_ERROR = 6,  // Internal error in the code. Should not happen ;-)
    STATUS_INVALID      = 7,  // Invalid argument.
    STATUS_CRC_WRONG    = 8,  // The CRC_A does not match
    STATUS_MIFARE_NACK    = 9   // A MIFARE PICC responded with NAK.
  };

  // A struct used for passing the UID of a PICC.
  typedef struct {
    byte    size;     // Number of bytes in the UID. 4, 7 or 10.
    byte    uidByte[10];
    byte    sak;      // The SAK (Select acknowledge) byte returned from the PICC after
     successful selection.
  } Uid;

  // A struct used for passing a MIFARE Crypto1 key
  typedef struct {
    byte    keyByte[MF_KEY_SIZE];
  } MIFARE_Key;

  // Member variables
  Uid uid;                // Used by PICC_ReadCardSerial().

  // Size of the MFRC522 FIFO
  static const byte FIFO_SIZE = 64;   // The FIFO is 64 bytes.

  /////////////////////////////////////////////////////////////////////////////////////
  // Functions for setting up the Arduino
  /////////////////////////////////////////////////////////////////////////////////////
  MFRC522(byte chipSelectPin, byte resetPowerDownPin);
  void setSPIConfig();

  /////////////////////////////////////////////////////////////////////////////////////
  // Basic interface functions for communicating with the MFRC522
  /////////////////////////////////////////////////////////////////////////////////////
  void PCD_WriteRegister(byte reg, byte value);
  void PCD_WriteRegister(byte reg, byte count, byte *values);
  byte PCD_ReadRegister(byte reg);
  void PCD_ReadRegister(byte reg, byte count, byte *values, byte rxAlign = 0);
  void setBitMask(unsigned char reg, unsigned char mask);
  void PCD_SetRegisterBitMask(byte reg, byte mask);
  void PCD_ClearRegisterBitMask(byte reg, byte mask);
  byte PCD_CalculateCRC(byte *data, byte length, byte *result);

  /////////////////////////////////////////////////////////////////////////////////////
  // Functions for manipulating the MFRC522
  /////////////////////////////////////////////////////////////////////////////////////
  void PCD_Init();
  void PCD_Reset();
  void PCD_AntennaOn();
  void PCD_AntennaOff();
  byte PCD_GetAntennaGain();
  void PCD_SetAntennaGain(byte mask);

  /////////////////////////////////////////////////////////////////////////////////////
  // Functions for communicating with PICCs
  /////////////////////////////////////////////////////////////////////////////////////
  byte PCD_TransceiveData(byte *sendData, byte sendLen, byte *backData, byte *backLen, byte *
     validBits = NULL, byte rxAlign = 0, bool checkCRC = false);
  byte PCD_CommunicateWithPICC(byte command, byte waitIRq, byte *sendData, byte sendLen, byte *
     backData = NULL, byte *backLen = NULL, byte *validBits = NULL, byte rxAlign = 0, bool
     checkCRC = false);

  byte PICC_RequestA(byte *bufferATQA, byte *bufferSize);
  byte PICC_WakeupA(byte *bufferATQA, byte *bufferSize);
  byte PICC_REQA_or_WUPA(byte command, byte *bufferATQA, byte *bufferSize);
  byte PICC_Select(Uid *uid, byte validBits = 0);
  byte PICC_HaltA();

  /////////////////////////////////////////////////////////////////////////////////////
  // Functions for communicating with MIFARE PICCs
  /////////////////////////////////////////////////////////////////////////////////////
  byte PCD_Authenticate(byte command, byte blockAddr, MIFARE_Key *key, Uid *uid);
  void PCD_StopCrypto1();
  byte MIFARE_Read(byte blockAddr, byte *buffer, byte *bufferSize);
  byte MIFARE_Write(byte blockAddr, byte *buffer, byte bufferSize);
  byte MIFARE_Decrement(byte blockAddr, long delta);
  byte MIFARE_Increment(byte blockAddr, long delta);
  byte MIFARE_Restore(byte blockAddr);
  byte MIFARE_Transfer(byte blockAddr);
  byte MIFARE_Ultralight_Write(byte page, byte *buffer, byte bufferSize);

  /////////////////////////////////////////////////////////////////////////////////////
  // Support functions
  /////////////////////////////////////////////////////////////////////////////////////
  byte PCD_MIFARE_Transceive( byte *sendData, byte sendLen, bool acceptTimeout = false);
  const char *GetStatusCodeName(byte code);
  byte PICC_GetType(byte sak);
  const char *PICC_GetTypeName(byte type);
  void PICC_DumpToSerial(Uid *uid);
  void PICC_DumpMifareClassicToSerial(Uid *uid, byte piccType, MIFARE_Key *key);
  void PICC_DumpMifareClassicSectorToSerial(Uid *uid, MIFARE_Key *key, byte sector);
  void PICC_DumpMifareUltralightToSerial();
```

```
329   void MIFARE_SetAccessBits(byte *accessBitBuffer, byte g0, byte g1, byte g2, byte g3);
330
331   ////////////////////////////////////////////////////////////////////////////////
332   // Convenience functions - does not add extra functionality
333   ////////////////////////////////////////////////////////////////////////////////
334   bool PICC_IsNewCardPresent();
335   bool PICC_ReadCardSerial();
336
337 private:
338   byte _chipSelectPin;     // Arduino pin connected to MFRC522's SPI slave select input (Pin 24,
        NSS, active low)
339   byte _resetPowerDownPin;  // Arduino pin connected to MFRC522's reset and power down input (
        Pin 6, NRSTPD, active low)
340   byte MIFARE_TwoStepHelper(byte command, byte blockAddr, long data);
341 };
342
343 #endif
```

RFID reader header file

### 8.2.3 LCD I²C header file

```
// ----------------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: Yes
//
// @file LiquidCrystal_I2C.h
// This file implements a basic liquid crystal library that comes as standard
// in the Arduino SDK but using an I2C IO extension board.
//
// @brief
// This is a basic implementation of the LiquidCrystal library of the
// Arduino SDK. The original library has been reworked in such a way that
// this class implements the all methods to command an LCD based
// on the Hitachi HD44780 and compatible chipsets using I2C extension
// backpacks such as the I2CLCDextraIO with the PCF8574* I2C IO Expander ASIC.
//
// The functionality provided by this class and its base class is identical
// to the original functionality of the Arduino LiquidCrystal library.
//
//
// @author F. Malpartida - fmalpartida@gmail.com
// ----------------------------------------------------------------------------
#ifndef LiquidCrystal_I2C_h
#define LiquidCrystal_I2C_h
#include <inttypes.h>
#include <Print.h>

#include "I2CIO.h"
#include "LCD.h"


class LiquidCrystal_I2C : public LCD
{
public:

   /*!
    @method
    @abstract   Class constructor.
    @discussion Initializes class variables and defines the I2C address of the
    LCD. The constructor does not initialize the LCD.

    @param      lcd_Addr[in] I2C address of the IO expansion module. For I2CLCDextraIO,
    the address can be configured using the on board jumpers.
    */
   LiquidCrystal_I2C (uint8_t lcd_Addr);
   // Constructor with backlight control
   LiquidCrystal_I2C (uint8_t lcd_Addr, uint8_t backlighPin, t_backlighPol pol);

   /*!
    @method
    @abstract   Class constructor.
    @discussion Initializes class variables and defines the I2C address of the
    LCD. The constructor does not initialize the LCD.

    @param      lcd_Addr[in] I2C address of the IO expansion module. For I2CLCDextraIO,
    the address can be configured using the on board jumpers.
    @param      En[in] LCD En (Enable) pin connected to the IO extender module
    @param      Rw[in] LCD Rw (Read/write) pin connected to the IO extender module
    @param      Rs[in] LCD Rs (Reset) pin connected to the IO extender module
    */
   LiquidCrystal_I2C( uint8_t lcd_Addr, uint8_t En, uint8_t Rw, uint8_t Rs);
   // Constructor with backlight control
   LiquidCrystal_I2C(uint8_t lcd_Addr, uint8_t En, uint8_t Rw, uint8_t Rs,
                     uint8_t backlighPin, t_backlighPol pol);

   /*!
    @method
    @abstract   Class constructor.
    @discussion Initializes class variables and defines the I2C address of the
    LCD. The constructor does not initialize the LCD.

    @param      lcd_Addr[in] I2C address of the IO expansion module. For I2CLCDextraIO,
    the address can be configured using the on board jumpers.
    @param      En[in] LCD En (Enable) pin connected to the IO extender module
    @param      Rw[in] LCD Rw (Read/write) pin connected to the IO extender module
    @param      Rs[in] LCD Rs (Reset) pin connected to the IO extender module
    @param      d4[in] LCD data 0 pin map on IO extender module
    @param      d5[in] LCD data 1 pin map on IO extender module
    @param      d6[in] LCD data 2 pin map on IO extender module
    @param      d7[in] LCD data 3 pin map on IO extender module
    */
   LiquidCrystal_I2C(uint8_t lcd_Addr, uint8_t En, uint8_t Rw, uint8_t Rs,
                     uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7 );
   // Constructor with backlight control
   LiquidCrystal_I2C(uint8_t lcd_Addr, uint8_t En, uint8_t Rw, uint8_t Rs,
                     uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7,
```

```
93                      uint8_t backlighPin, t_backlighPol pol);
    /*!
95   @function
     @abstract   LCD initialization and associated HW.
97   @discussion Initializes the LCD to a given size (col, row). This methods
     initializes the LCD, therefore, it MUST be called prior to using any other
99   method from this class or parent class.

101  The begin method can be overloaded if necessary to initialize any HW that
     is implemented by a library and can't be done during construction, here
103  we use the Wire class.

105  @param      cols[in] the number of columns that the display has
     @param      rows[in] the number of rows that the display has
107  @param      charsize[in] size of the characters of the LCD: LCD_5x8DOTS or
     LCD_5x10DOTS.
109  */
    virtual void begin(uint8_t cols, uint8_t rows, uint8_t charsize = LCD_5x8DOTS);
111
    /*!
113  @function
     @abstract   Send a particular value to the LCD.
115  @discussion Sends a particular value to the LCD for writing to the LCD or
     as an LCD command.
117
     Users should never call this method.
119
     @param      value[in] Value to send to the LCD.
121  @param      mode[in] DATA - write to the LCD CGRAM, COMMAND - write a
     command to the LCD.
123  */
    virtual void send(uint8_t value, uint8_t mode);
125
    /*!
127  @function
     @abstract   Sets the pin to control the backlight.
129  @discussion Sets the pin in the device to control the backlight. This device
     doesn't support dimming backlight capability.
131
     @param      0: backlight off, 1..255: backlight on.
133  */
    void setBacklightPin ( uint8_t value, t_backlighPol pol );
135
    /*!
137  @function
     @abstract   Switch-on/off the LCD backlight.
139  @discussion Switch-on/off the LCD backlight.
     The setBacklightPin has to be called before setting the backlight for
141  this method to work. @see setBacklightPin.

143  @param      value: backlight mode (HIGH|LOW)
     */
145  void setBacklight ( uint8_t value );

147 private:

149  /*!
     @method
151  @abstract   Initializes the LCD class
     @discussion Initializes the LCD class and IO expansion module.
153  */
    int  init();
155
    /*!
157  @function
     @abstract   Initialises class private variables
159  @discussion This is the class single point for initialising private variables.

161  @param      lcd_Addr[in] I2C address of the IO expansion module. For I2CLCDextraIO,
     the address can be configured using the on board jumpers.
163  @param      En[in] LCD En (Enable) pin connected to the IO extender module
     @param      Rw[in] LCD Rw (Read/write) pin connected to the IO extender module
165  @param      Rs[in] LCD Rs (Reset) pin connected to the IO extender module
     @param      d4[in] LCD data 0 pin map on IO extender module
167  @param      d5[in] LCD data 1 pin map on IO extender module
     @param      d6[in] LCD data 2 pin map on IO extender module
169  @param      d7[in] LCD data 3 pin map on IO extender module
     */
171  void config (uint8_t lcd_Addr, uint8_t En, uint8_t Rw, uint8_t Rs,
                 uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7 );
173
    /*!
175  @method
     @abstract   Writes an 4 bit value to the LCD.
177  @discussion Writes 4 bits (the least significant) to the LCD control data lines.
     @param      value[in] Value to write to the LCD
179  @param      more[in]  Value to distinguish between command and data.
     COMMAND == command, DATA == data.
181  */
    void write4bits(uint8_t value, uint8_t mode);
183
    /*!
185  @method
     @abstract   Pulse the LCD enable line (En).
187  @discussion Sends a pulse of 1 uS to the Enable pin to execute an command
```

```
     or write operation.
189  */
    void pulseEnable(uint8_t);
191

193     uint8_t _Addr;              // I2C Address of the IO expander
        uint8_t _backlightPinMask; // Backlight IO pin mask
195     uint8_t _backlightStsMask; // Backlight status mask
        I2CIO   _i2cio;             // I2CIO PCF8574* expansion module driver I2CLCDextraIO
197     uint8_t _En;                // LCD expander word for enable pin
        uint8_t _Rw;                // LCD expander word for R/W pin
199     uint8_t _Rs;                // LCD expander word for Register Select pin
        uint8_t _data_pins[4];      // LCD data lines
201
};
203
#endif
```

LCD I2C header file