Smart Lock System

CONTENT

1 INTRODUCTION

In this work we will be create an assignment for SMVIT course and try to come up with a solution in accordance with 7D methodology. We will specify project goals, make an analysis of current SW/HW solutions in the chosen field, propose a solution, design the solution, implement the solution, describe the solution and finally evaluate it. Since generally, "we" is used as personal pronoun even for single-author papers, I will resort to using pronoun "we" even though this paper has a single author.

1.1 Project goals

Nowadays we have access to broad spectrum of electronic devices designed for integration with other such devices that allow us to build complex solutions specifically tailored for certain tasks. Goal of this project is to design and construct such a device. Simultaneously, we want for our solution to have some practical usage in a household environment. Access control to some physical location will be always necessary in all kinds of environments from ordinary households to enterprise business environments. Therefore, we decided to design and construct such a system for a household use. Functionality will be demonstrated on one access point (a safe). It is important to distinguish what this project is and what it is not. Our goal is not to provide a complex commercial solution, but to construct a low-cost multi-input safe prototype for a personal use and to deepen our knowledge of embedded systems and NFC technologies.

System requirements:

1. Password-based access control
2. "RFID card"-based access control
3. RFID card management
4. Password management
5. Ability to inform about system state through auditory and visual cues
6. Ability to inform about system state through LCD screen

1.2 Assignment

Basic information:
Course: **SMVIT**
Author: **Bc. Dávid Csomor**
Year: **2020/2021**
Name of the Solution: **Smart Lock System**

Assignment:
Design, construct, implement and verify a low-cost, multi-input smart safe prototype. Content in the locked location should be accessible after providing a correct RFID card or after inputting a specific password chosen by the user. System will provide information about its state through LCD screen.

Solution will contain:
1. Proper analysis
2. Description of proposed solution
3. Resulting artifacts and conclusion
4. Used literature
5. Technical documentation

2 ANALYSIS

In this section, we will provide an analysis of existing solutions including the integrated development environments (IDEs).

2.1 Analysis of existing solutions

Most businesses, facilities and various other institutions are controlling access to its buildings and assets. Most of the time it is done through classic mechanical lock and a corresponding key. In our proposed solution we will demonstrate access control through "digital" means on a safe. Later, it should be possible to extend this solution with logging and some kind of advanced access management mechanism as well. Servo motor, which will control the lock, can be substituted with magnetic or other kind of electronic locking mechanism. Nowadays we can find similar simple solutions in Chinese online stores (like Ebay, Aliexpress and others) starting at the price of 14 EUR depending on the model and capabilities.

Since it is already so cheap and a widely available (not necessarily used) solution, it is unlikely that we will produce a considerably improved or cheaper solution as a result of a student course assignment. During the design phase we will try to come up with a cheapest possible way to construct this safe while still meeting the requirements and intended functionality. There are only few options to choose from when picking an LCD screen and a keyboard. Therefore, these are not going to be discussed in the analysis section.

2.1.1 Computing power

During the process of choosing the right processing HW, which will "contain" and execute a desired logical routine, we took into account the following three devices:

1. Raspberry Pi 3 (*RP3*)
2. Arduino UNO Rev3
3. Arduino Mega2560 Rev3

Following overview will be a simplified one because the goal is not to provide a comprehensive low-level comparison, but a quick summary of the most important features. First, we would like to point out differences between RP3 and Arduino devices. The main difference is the fact that in their core they are a completely different device. While Arduino is a microcontroller into which we load our "simple" (and relatively small program) and then it is run in the loop, RP3 is a single-board microcomputer. RP3 is much richer in terms of features and basically outperforms Arduino in every aspect, except for the presence of analogue female connectors and power consumption. For the purposes of our project assignment, we have decided to pick an Arduino based embedded board because of its lower power consumption, lower price and its satisfactory computing capabilities.

Next step was to decide between Arduino UNO and Arduino MEGA2560. Main difference between these two boards and microcontrollers is in the number of available pins and memory. Due to our goal of minimizing the financial cost we have decided to purchase an Arduino Uno which should have sufficient number of pins and processing power for the required parts.

2.1.2 RFID technology

We have a broad spectrum of diverse types of RFID (*Radio Frequency IDentification*) readers and cards, which work on different frequencies and have various communication interfaces. Based on their working frequency we can divide RFID equipment into following categories:

- Low frequency range – 30 to 300 kHz (LF – Low Frequency)
- High frequency – 13,56 MHz (HF – High Frequency)
- Ultra-High frequency range – 300 to 3000 MHz (UHF – Ultra High Frequency)

Of course there are various other technologies depending on the usage of active or passive transmitting (with or without source respectively), but we will be considering mainly the technology working on the 13,56 MHz frequency since it is widely used as a means of access control in various environments. Working frequency has a considerable impact on the distance at which the reader is able to read information stored on the card.

An RFID system consists of two main components. Tag (transponder) located at the object we want to be identified and a reader. Reader consist of a control unit, radio frequency module and an antenna coil which generates high frequency electromagnetic field (EMF). On the other side of this "transaction" there is a tag/card mentioned previously (it is usually a passive component). It consists of antenna and an electronic microchip, so when it gets near the EMF of the receiver, power is generated (due to induction) and it powers the chip on the tag. Figure 2-1 depicts working of such a mechanism.

There are two main methods of transmitting data this way. One is called "load manipulation". It is done by switching on and off at the antenna which can be measured as voltage drop. This change of voltage can be interpreted as 0 and 1 and can be used to wirelessly transfer the data. Other mechanism is called "backscattered coupling". In this method, the tag uses part of the received power for generating another EMF which is recognized by an antenna in the reader and data is transmitted through different means.

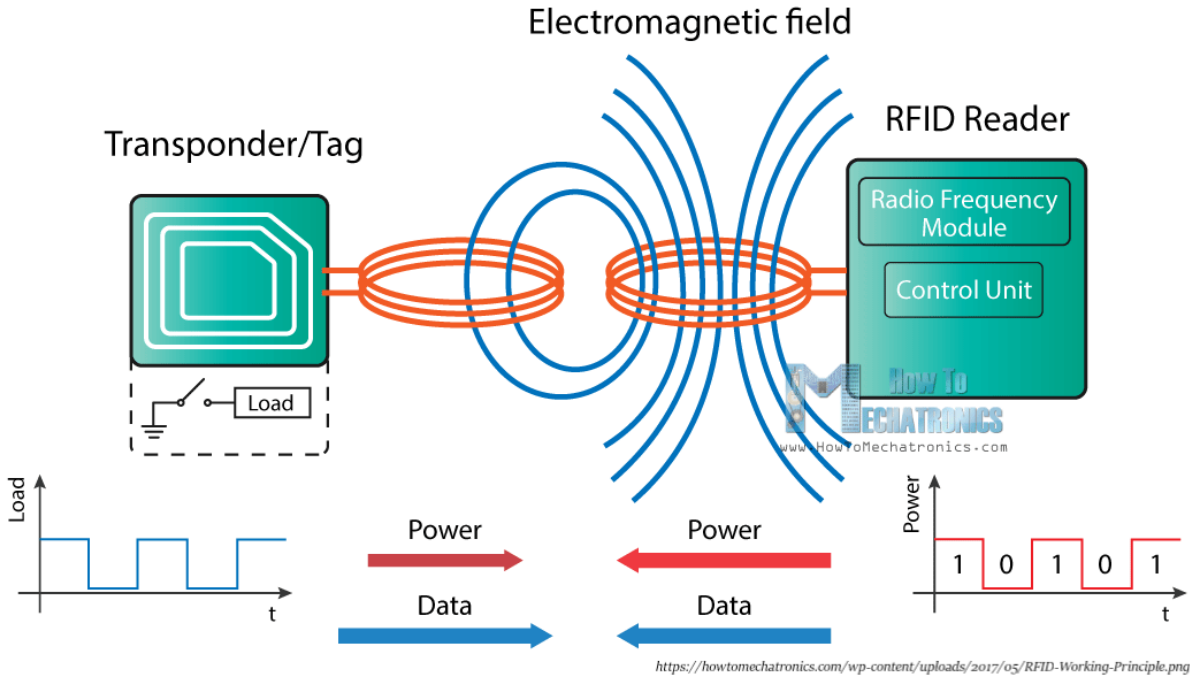


Figure 2-1 RFID system data transmission method.

Highest range is attributed to Ultra High Frequency readers with reading distance ranging from 25 to 100+ meters depending on the exact implementation and surrounding conditions. These systems have only been around since the mid 1990s and countries around the globe are yet to agree on a single specific spectrum for this equipment. Different countries have different bandwidth and power restrictions for these systems. For example European Union has UHF RFID range varying from 865 to 868 MHz with readers being able to transmit with maximum power of 2 Watts of ERP (*Effective Radiated Power*), while devices in North America are operating at a frequency range from 902 to 928 MHz and ERP of only 1 Watt.

Cards with a shortest range frequency communications (LF) have similarly their own advantages and disadvantages, but the common denominator is higher overall production cost (when compared to HF), which might be a viable solution when a business have financial resources to spare or the other ranges (and their underlying technology) don't meet their requirements. Their reading distance is between 0 and 10 cm. Theoretically we could use the LF equipment, but this would not be as cost-effective as it would be to use a High Frequency equipment, not mentioning the aspect of a weaker security measures present in the LF equipment. LF equipment is viable option to use in ID chips for pets' collars or when there is no need for any kind of security measures.

High Frequency equipment seems ideal for our purposes. These tags/cards have 1kB (or 4kB depending on the model) of memory and are read/write capable. They have a microchip that is able to do arithmetic operations and is compliant with ISO-14443 type 'A' protocol. The MIFARE family has a wide range of products covering various different requirements and applications. Average tag production cost is between 0,2 EUR and 10 EUR. Reading distance is between 0 and 30 centimeters. They can be commonly found in membership cards, store value cards, access cards, etc., but most importantly they can be

used and simulated with NFC technology (already present in numerous mobile phones and other devices). We will be using readers and cards working on a 13,56 MHz frequency with MIFARE technology. These are recommended to use when we need more secure solutions since they allow to take advantage of secure authentication protocols. In 2017, company IDentiv launched their high-frequency 13,56 MHz cards in response to security weaknesses present in 125kHz proximity card systems. Figure 2-2 depicts such a MIFARE card with red antenna connected to the chip in the middle. Information stored on the card can be read with any cell phone with NFC capabilities.

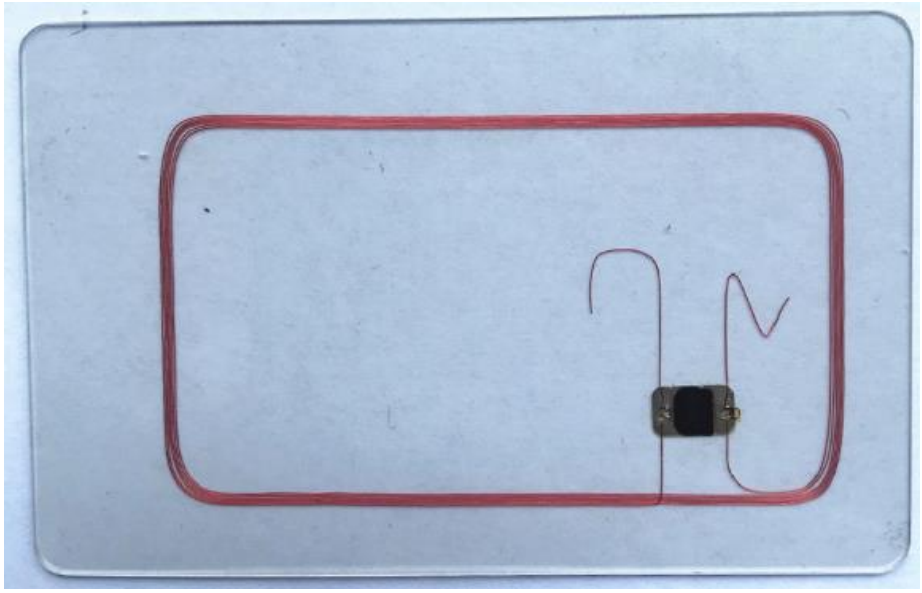


Figure 2-2 Mifare Classic 1kB 13,56MHz card.

Finally, we need to do an analysis of available card readers. There are readers which have a built-in memory to store card IDs, operating logic and other kinds of information. This role will be covered by Arduino and its EEPROM. As for the reader module itself, we did not find any viable alternative for the card reader used in this work.

2.2 Integrated Development Environment (IDE)

Before the development we settled on trying out these IDEs. Namely:

1) Eclipse 2019-09 R (4.13.0) + Eclipse C++ IDE for Arduino 3.0 + Arduino Eclipse IDE named Sloeber v4.3.1
2) PlatformIO v4.1.1b5 + MS Visual Studio Code v1.40
3) Arduino IDE v1.8.13

At first, we tried using Eclipse IDE, which was quite outdated and problematic to use. To be able to use it, we had to first install all the mentioned plugins. Their installation was only partially successful and in the end, it was a substantial problem to even create (or supply an existing) *.cpp file. Such an attempt always led to a serious error message and an exception. We tried even older Eclipse versions, but were ultimately unsuccessful and wasted one whole day trying setup this environment before finally moving on to option number 2.

PlatformIO was our second option and it is an open-source ecosystem designed for development of IoT devices and their programming. It supports multiple well-established text editors/development environments like MS-VS-Code, Atom and others. This IDE worked much better than Eclipse unless we tried to include literally any library. We were able to download libraries and repositories for Arduino, but their insertion/linkage to code was extremely complicated and had a quite steep learning curve. Typical way of including a library in C language is done through insertion of <xxxx.h> header and putting necessary libraries somewhere the compiler could find them, but It was unnecessarily complicated here. Main problem was that, at the time of implementing our solution, there was no comprehensive and easy to follow guide which would show beginners how to accomplish such a simple task. Also managing such libraries (adding and removing them) is quite complicated. It can be done, but it would take too much time (to learn – figure out) because the existing guides are quite inconsistent and we needed some simple solution which would allow us to easily add and remove different versions of various libraries (since some newer library versions are not always compatible with certain hardware). Mentioned library management problems resulted in huge number of compilation warnings, errors and failed compilation. Big advantage of this IDE extension was "IntelliSense" (*intelligent code completion*), which provides autocomplete capabilities, but since this environment was not satisfying to use, we moved on to option number 3.

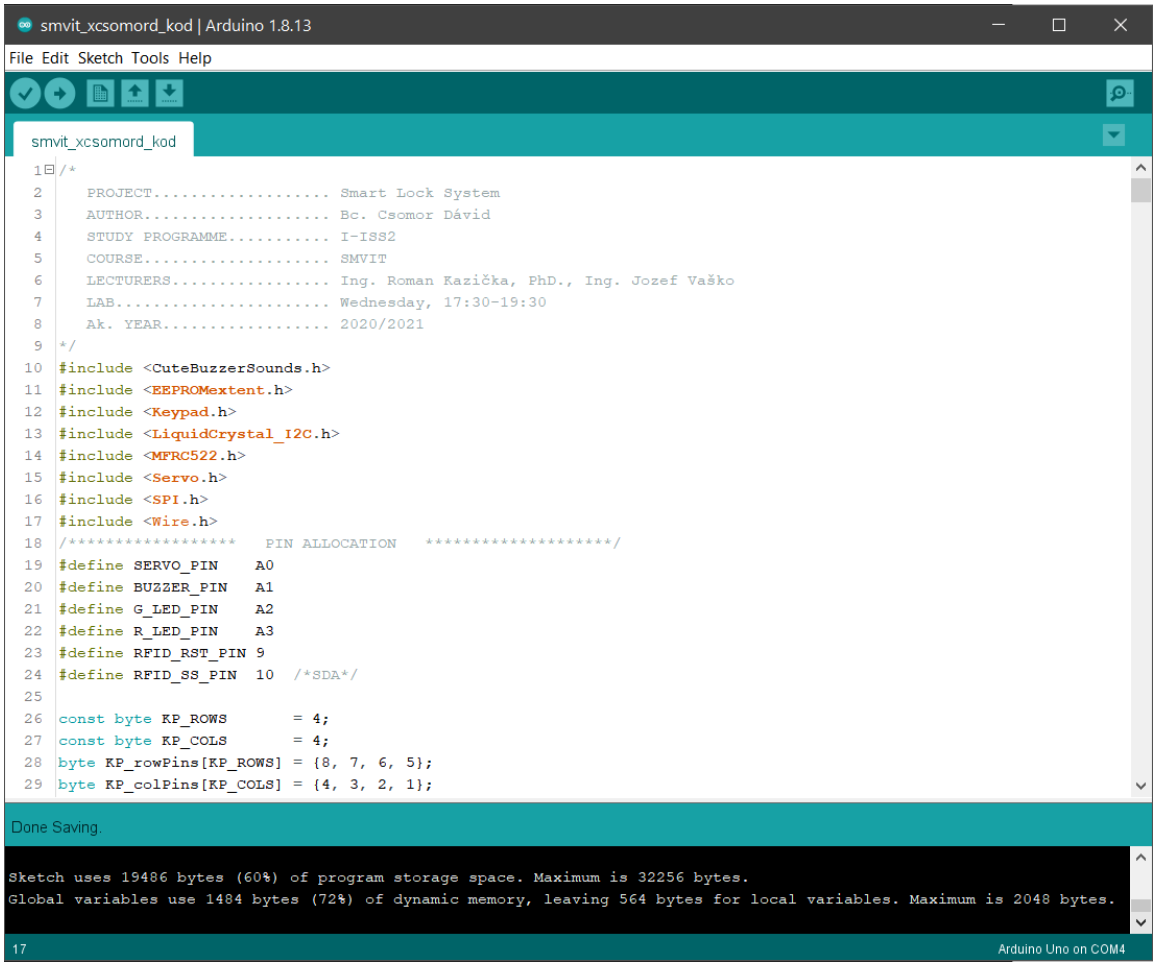


Figure 2-3 Preview of development environment Arduino IDE (1.8.13).

Arduino IDE was the last in our list environments to try and ended up being the one we used. It claims to be designed for people with little to no prior experience with programming. We cannot completely agree with this statement since the absence of autocomplete can be a bit problematic upon exceeding a certain code length and when using functions (without a preview of function parameter list). On the other hand, it allows to effortlessly find, install and insert various (community driven) libraries as depicted on Figure 2-5. It is a code editor with basic features like syntax highlighting, code auto format, section alignment function, etc. Arduino IDE can compile and upload code to a device attached through USB port with a single press of a button, which can be extremely useful for beginners. Figure 2-3 depicts a preview of Arduino IDE.

```
1  void setup() {
2    // put your setup code here, to run once:
3
4  }
5
6  void loop() {
7    // put your main code here, to run repeatedly:
8
9  }
```

Figure 2-4 Preview of main Arduino functions.

Programs for Arduino are written in C and C++. Figure 2-4 depicts basic Arduino functions, which must be implemented in order for Arduino to function properly. Function ***setup()*** is launched only once at the beginning of the program (after the Arduino is powered on) and is used to setup communication, initialize variables and call a main function. ***Loop()*** function is first called from setup() and it is repeatedly executed until the Arduino restarts or powers off.

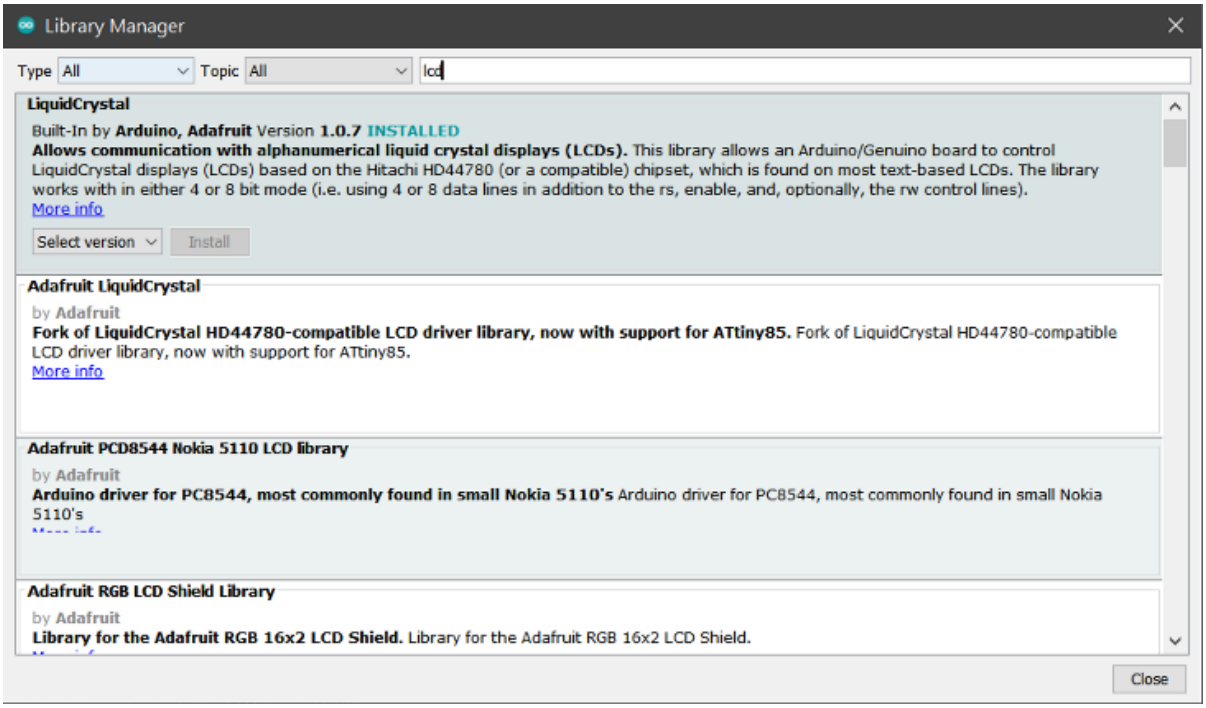


Figure 2-5 Arduino IDE Library Manager.

To construct our circuit board located later in this work we used a (open source) program called Fritzing v.0.9.3, which is designed to allow everybody to design, build and test electronic circuits. We can label it as an IDE. It was used to design, test and verify our solution. Figure 2-6 depicts Fritzing "IDE", which is an amateur level CAD software intended for design of electronics HW, prototypes and even PCBs.

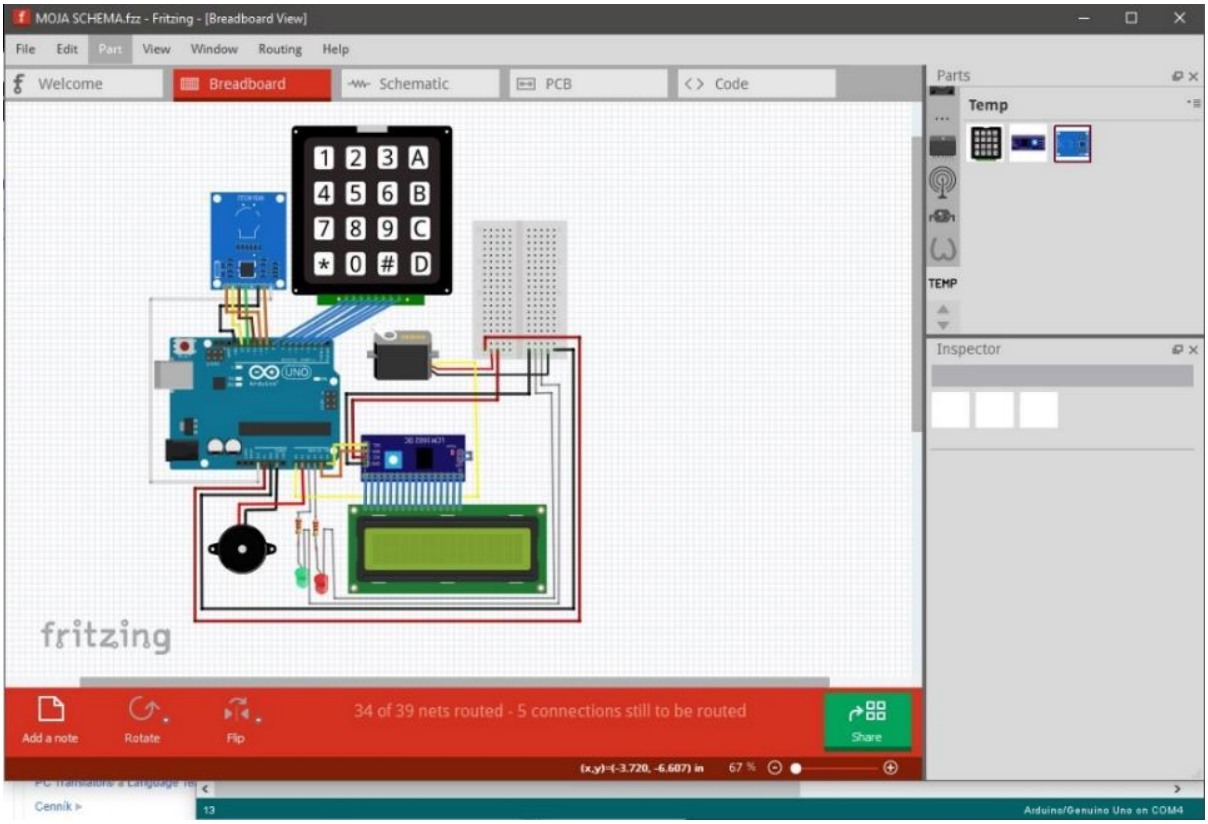


Figure 2-6 Preview of development environment Fritzing.

3 SOLUTION DESCRIPTION

This section covers a solution proposal, list of components used in the project, circuit and breadboard schematics, state diagram, more thorough description of the prototype and finally a manual.

3.1 Solution proposal

After analysis we composed a circuit prototype. Figure 3-1 depicts a circuit diagram of the solution It shows components, which are used and the way they are connected. This diagram (as well as others) is updated and reflects the current state of the design. The display was originally connected without an I^2C module and was occupying too many connectors, which resulted in an inability to use LED diodes and keys 'A', 'B', 'C', 'D'. Initial solution was equipped with separate contrast regulation mechanism (for an LCD screen), but it was removed due to the added converter module having its own contrast regulation mechanism.

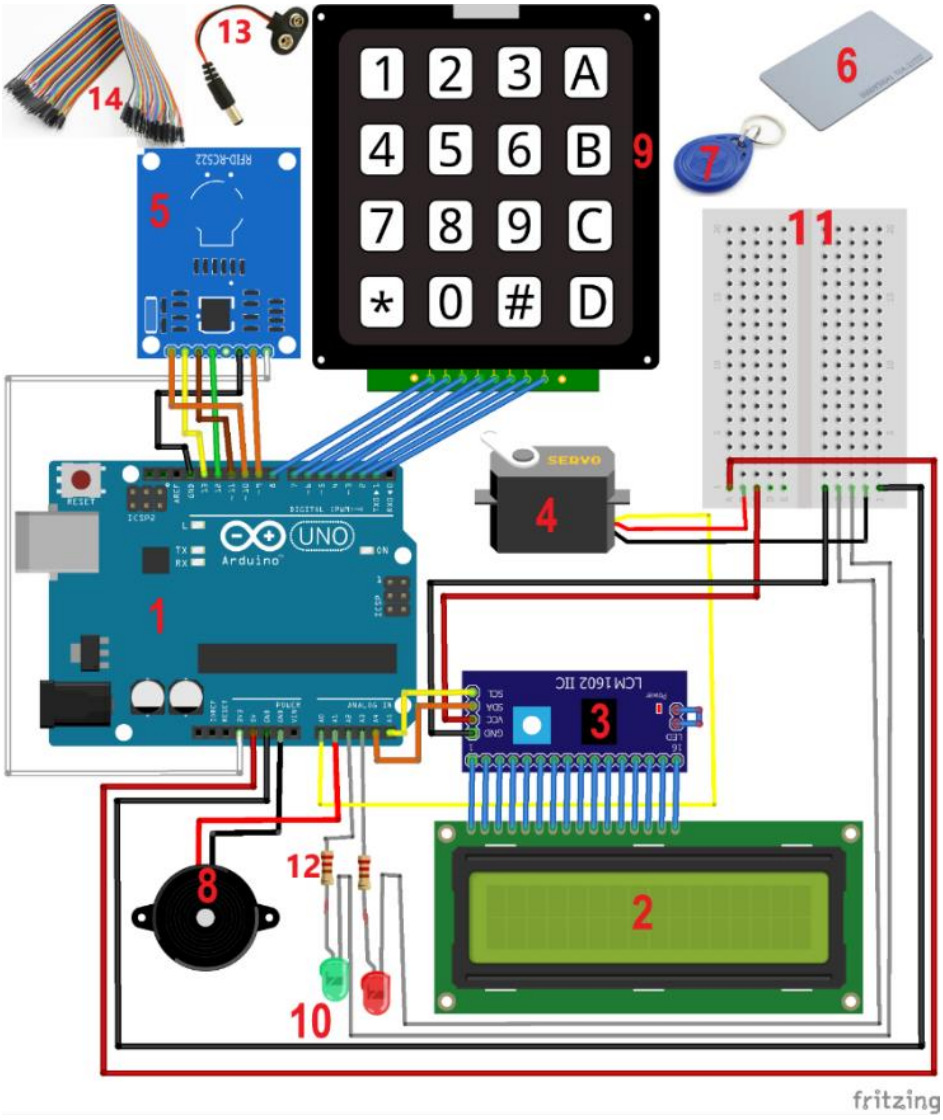


Figure 3-1 Components used and their connection.

3.2 Components

List of used components:

1) **(1)** Arduino UNO Rev3 (1x)
2) **(2)** LCD1602 Module (1x)
3) **(3)** Converter I2C for LCD display (1x)
4) **(4)** Servomotor SG90 Micro Servo (1x)
5) **(5)** MFRC522 module (1x)
6) **(6)** RFID card S50 (13,56MHz) (3x)
7) **(7)** RFID chip Sebury (13,56MHz) (1x)
8) **(8)** Active buzzer (1x)
9) **(9)** Membrane keyboard 4x4 (1x)
10) **(10)** LED 3MM red (1x)
11) **(10)** LED 3MM green (1x)
12) **(11)** Breadboard (17x8) (1x)
13) **(12)** Resistor (220 ohm) (2x)
14) **(13)** Battery connector 9V-5,5 / 2,1 (1x)
15) **(14)** Dupont wires M/M, F/M
16) USB cable 2.0 type A/B (1x)

3.2.1 Arduino UNO (Rev3) [1]

Figure 3-2 Arduino UNO (Rev3).

Arduino Uno rev3, depicted on Figure 3-2, is a microcontroller board based on the ATmega328P from Atmel. It has 14 digital input/output pins (6 can be used as PWM outputs) and additional 6 analog inputs. Its working frequency is 16 MHz, its able to transmit data through USB 2.0 (through serial port converter), it has an ICSP interface designed for programming AVR microcontrollers, power connector and a button which is able to restart the device.

Component specification:

- Microcontroller ATmega328
- Operating voltage ..... 5 V
- Input voltage ..... 7 – 12 V (max 6 – 20 V)

- Digital I/O pins ..... 14 *(6 provide PWM output, 0 - 255 values)*
- Analog input pins ..... 6 *(A0-A5, values:0 - 1024)*
- SRAM ..... 2 kB *(ATmega328)*
- Flash memory ..... 32 kB *(ATmega328 – from which 0,5kB is for bootloader)*
- EEPROM ..... 1kB *(ATmega328)*
- Dimensions ..... 68,6 × 53,3 mm
- Weight ..... 25 g

3.2.2 LCD - LCM1602A Module

Figure 3-3 LCD - LCM1602A Modul.

Blue LCD screen LCM1602A, depicted on Figure 3-3, is a displaying module with built-in white LED backlight consisting of 16 columns and 2 rows. There are two different types of LCDs available. Graphical and character LCDs. LCM1602A is a character LCD. The industry standard HD44780 LCD displays require by default 4 or 8 parallel data lines. These modules can work in either of those modes, namely 8-bit and 4-bit mode. Figure 3-4 depicts such an LCD screen without the use of I^2C converter and is connected in a 4-bit mode.

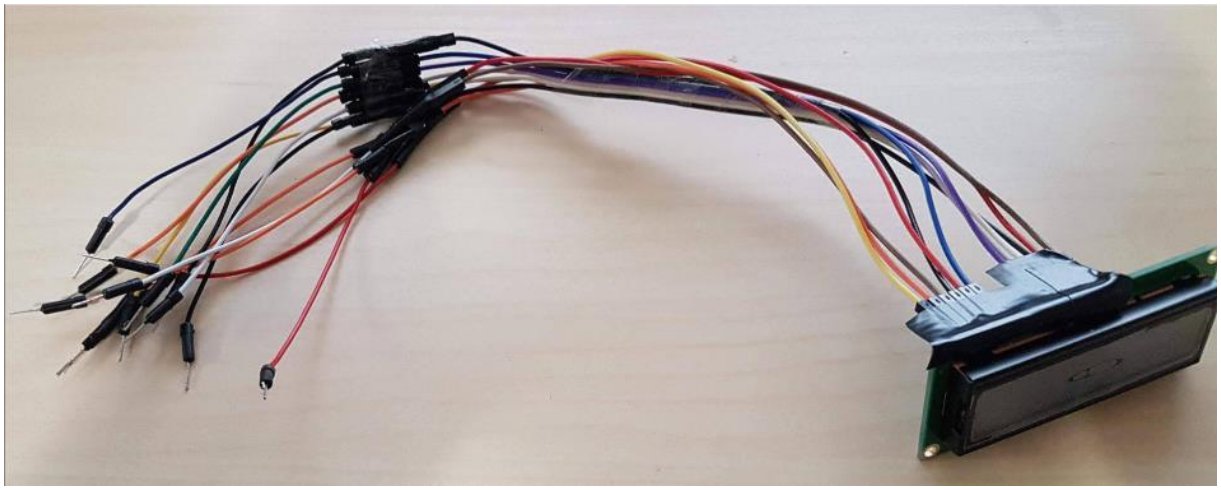


Figure 3-4 LCD screen without I2C converter.

After adding and I^2C converter all of these pins were later replaced with only 4 pins (in total) for the whole LCD. Figure 3-5 describes every PIN on the LCD screen. Main difference between the 4-bit and the 8-bit mode is in the number of PINs needed to communicate and in the latency, meaning the number of cycles it requires to transmit the

same amount of data. 8-bit mode can transmit one character during 1 cycle (1 ASCII sign = 8 bits), while the 4-bit mode needs 2 cycles to transmit one character and uses only pins DB4 to DB7. However, there is a workaround in the form of I^2C converter module, which can be plugged between the LCD screen and the Arduino. It allows the LCD to communicate through Arduinos I^2C bus and significantly reduce the number of used pins along with the possibility to remove a 10K potentiometer needed for LCD contrast adjustment. Figure 3-6 depicts this potentiometer, which was used in earlier stages of this project, but was later replaced with I^2C converter. LCD allows to display 16 character per line with 2 lines, so 32 characters total. One character can be constructed through a 5x8 grid of fine dots with dimensions of 0,55 x 0,5 mm.

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

http://www.datasheet-pdf.com/PDF/LCM1602A-Datasheet-LONGTECHOPTICS-866417

Figure 3-5 PIN interface description for LCM1602A.

Component specification:

- Type ..... LCD Module with liquid crystals
- Controller ..... type HD44780
- Operating voltage..... 5 V
- Operating temperature ..... 0 – 50°C
- Dimensions ..... 80 x 36 x 1,6 mm
- Weight ..... 50 g

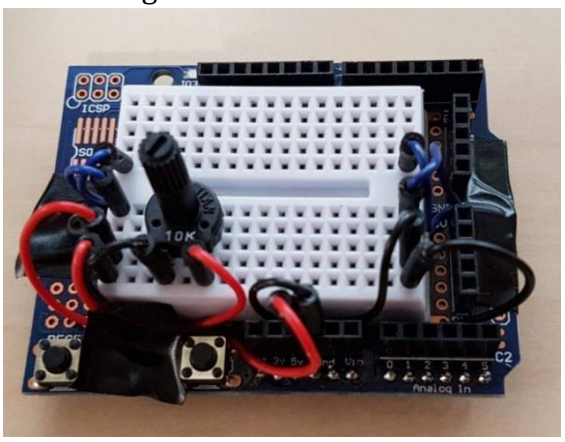


Figure 3-6 10k potentiometer used for contrast regulation.

3.2.3 I²C converter for LCD

Figure 3-7 I²C converter for LCD.

I²C (*Inter-Integrated Circuit*) converter for LCD, depicted on Figure 3-7, is a "simple" converter which allows Arduino to communicate with a (16x2 and 20x4) LCD screen through I²C bus and protocol. Main functionality is taken care of by I/O expander PCF8574 depicted on Figure 3-8. It connects through 4 PINS: GND, VCC, SDA and SCL. Figure 3-9 depicts this converter along with a description of its main parts. On the rear side of the board there is a pre-soldered 10k potentiometer (blue box with grey opening), which can manually regulate contrast and a mechanical jumper to control the LCDs contrast (black jumper on the left).

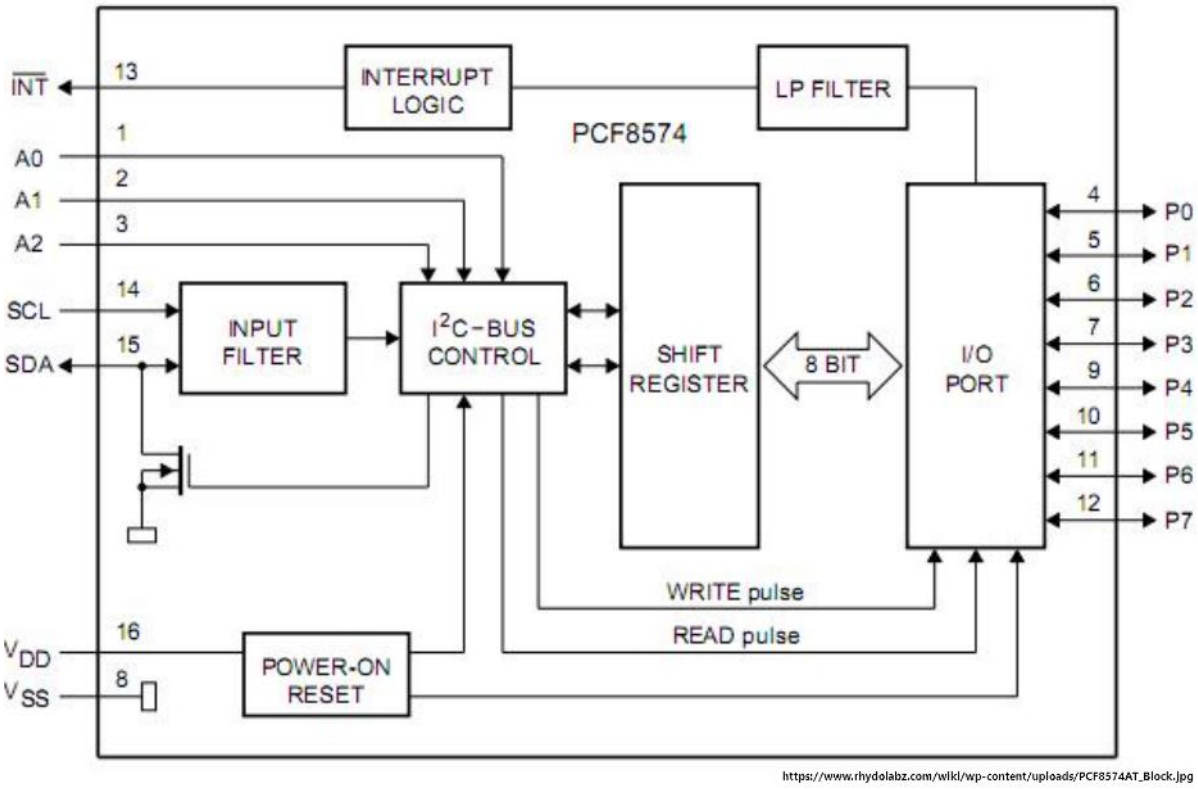


Figure 3-8 Block scheme for remote 8-bit I/O expander PCF8574. [2]

Circuit consists of 8-bit bi-directional port and an interface for an I²C bus. It is also able to service interrupts (INT) directly from Arduinos microcontroller. Main function of this converter is to convert data sent through an I²C bus to (and from) parallel bus/format. We can also label it as a certain kind of (de)multiplexing device. It has three 1 bit inputs (A0, A1, A2), which are used to set its physical address. We can easily calculate the highest number of devices connected to the I²C bus by solving the equation 2^3. Usually it communicates on address 0x27 (or 0x3F if it has a newer model of PCF8574A

expander), but theoretically the communication can occur on any address from 0x20 to 0x27 (or 0x38 to 0x3F respectively).

Figure 3-9 I^2C converter description.

Component specification:

- V/V expander PCF8574
- Working voltage ..... 5 V
- Working current ..... 30 mA
- Working temperature ..... -20 – 70°C
- Dimensions ..... 87 x 32 x 13 mm

I^2C interface

Default I^2C address is 0x27 and it should be marked on the "main" board of the device. This address can be changed by soldering jumpers to the board. It is a synchronous, multi-master, multi-slave, packet switched communication bus. It is widely used to attach lower-speed peripheral integrated circuits to processors and microcontrollers. The I^2C is developed to overcome the difficulties faced when transmitting data with the help of other communication protocols such as UART and SPI.

Communication is through SCL (*Serial Clock Line*) and SDA (*Serial Data Line*) and uses a 7-bit addressing broadcasted by device. Previously mentioned clock signal synchronizes data transfer between the device and an I^2C bus. Alternative for this way of communication is a so-called TWI (*Two Wire interface*) developed by Atmel (and its not a protected by trademark like the I^2C).

3.2.4 TowerPro Servomotor SG90

Figure 3-10 TowerPro Servomotor SG90.

TowerPro SG90 9g Mini Servo, depicted on Figure 3-10, is servo with ability to rotate 180° ideal for simulating the locking mechanism on the safe.

Component specification:

- Operating speed ..... 0,10s / 60°
- Torque (4,8V) 1,2 Kg/cm2
- Torque (6,6V) 1,6 Kg/cm2
- Rotation ..... 180°
- Working voltage..... 3 – 7,2 V (typically 5 V)
- Working Temperature ..... -30 – 60°C

3.2.5 Module RFID reader MFRC522

Figure 3-11 Module RFID reader MFRC522.

RFID reader, depicted on Figure 3-11, is equipped with integrated MFRC522 circuit and to communicate with a tag/card/responder it uses wireless NFC (*Near Field Communication*) technology through some of the following interfaces: UART, SPI or I^2C. This module is set to communicate through SPI interface by default (without any additional modifications). Figure 3-12 depicts such a reading module and describes its PINs. SPI protocol (and therefore our reader) communicates through SS, SCK, MOSI, MISO. Addressing is done through an SS PIN (during logical 0, sending and receiving is active). It uses clock signal to regulate data transfer. It is of 1 master, multiple slaves, full-duplex nature.

Component specification:

- Relative env. humidity..... 5% – 95%
- Frequency ..... 13,56 MHz
- Data rate(max) ..... 10 Mbit/s
- Working voltage ..... 2,5 – 3,6 V (typically 3,3 V)
- Working current ..... 13 – 26 mA
- Working current (idle)..... 10 – 13 mA
- Working temperature ..... -20 – 80°C
- Dimensions ..... 60 x 40 x 5 mm
- Weight ..... 30 g
- Supported cards..... Mifare1 S50, Mifare1 S70, Mifare UltraLight, Mifare Pro, Mifare Desfire

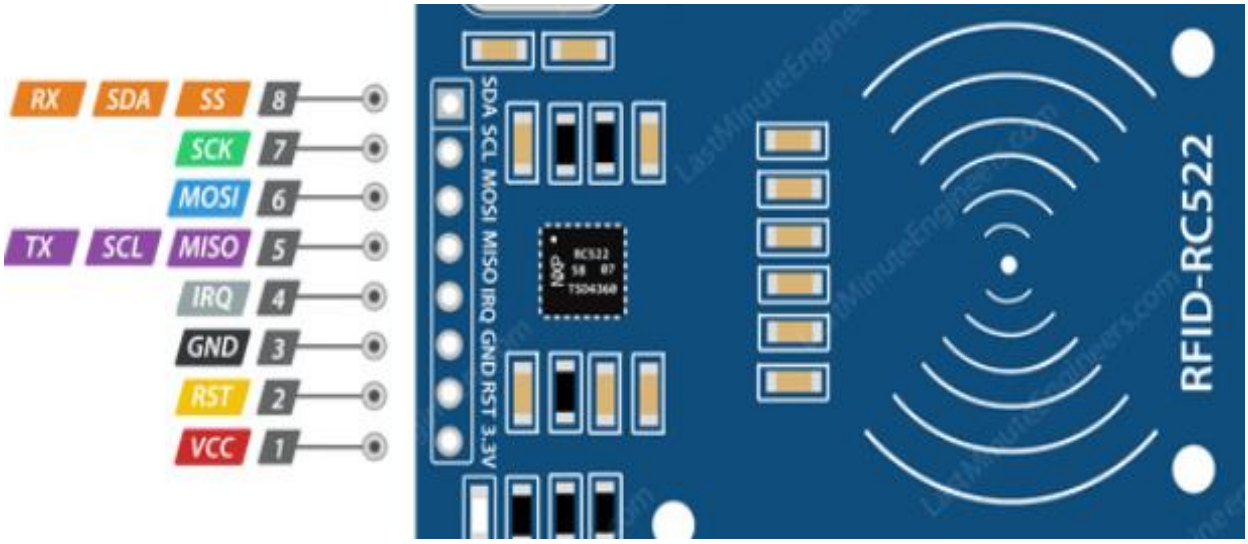


Figure 3-12 RFID reader MFRC522 PIN description.

3.2.6 MF1 IC S50 RFID card

Figure 3-13 MF1 IC S50 RFID card.

Wireless chip card Mifare S50 1kB, depicted on the left side of Figure 3-13, is the most common type of card used (in the area of access control) it works on the frequency of 13,56 MHz and contains 1 kB of memory with 752 bytes of writable memory. Every single card has its unique UID identifier, which should uniquely identify any given card. Since the UID space is finite and we use cards with 4-byte identifiers, there are approximately 3,7 billion unique IDs.

Component specification:

- Frequency ..... 13,56 MHz
- Working temperature ..... -40 – 80°C
- Type ..... Mifare 1
- EEPROM size 1kB
- Internal division ..... 16 sectors
- Num. of writes to EEPROM ..... 100 000 cycles
- ISO standard ..... ISO 14443 / 14443 A
- Transmission range ..... 0 ~ 10 cm

3.2.7 Active buzzer (Mini Piezo Buzzer)

Figure 3-14 Active buzzer (Mini Piezo Buzzer).

Active buzzer, depicted on Figure 3-14, generates tone using an internal oscillator requiring DC voltage. Passive buzzer does not have an internal oscillator and it needs additional signal source that supplies a sound signal.

Component specification:

- Working voltage ..... 3,5V - 5,5 V
- Working current ..... <25 mA
- Frequency ..... 2300 ± 500 Hz
- Diameter ..... 12 mm
- Height ..... 9,5 mm

3.2.8 Membrane keyboard 4x4

Figure 3-15 Membrane keyboard 4x4.

Membrane keyboard 4 x 4 AVR (16 keys), depicted on Figure 3-15, is suitable as an input control panel for our safe. Its big advantage is its extremely low profile and ability to be effortlessly mounted to any surface.

Component specification:

- Working voltage(max) ..... 24 V
- Working current (max) ..... 30 mA
- Working temperature ..... 0 – 50°C
- Dimensions (keyboard) ..... 69 x 76 mm
- Dimensions (cable) ..... 20 x 88 mm

3.2.9 LED 3MM

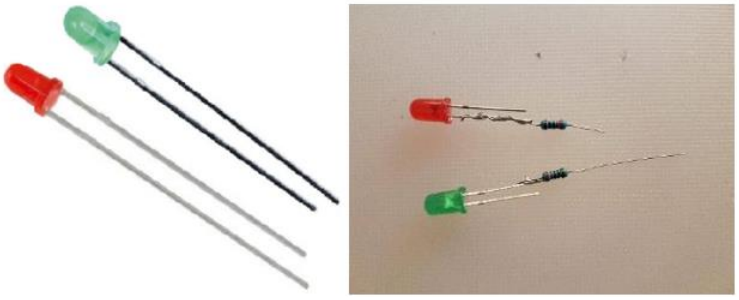


Figure 3-16 LED 3MM.

LED diode, depicted on Figure 3-16, is a semiconductor light source that emits light when current flows through it. On the picture, we have already soldered a (220 ohm) resistor to it in order to prevent damaging the LED in the long run. It might work without the resistor for a while, but would gradually wear down the LED and cause the LED to be dimmer or stop working altogether.

Component specification:

- Working voltage ..... 2,5 – 4 V
- Working voltage ..... 10 – 25m A

3.3 Circuit schematics

Next important part is the schematic view of the breadboard layout depicted on Figure 3-17. This scheme is just a different perspective of the breadboard layout presented earlier in this work.

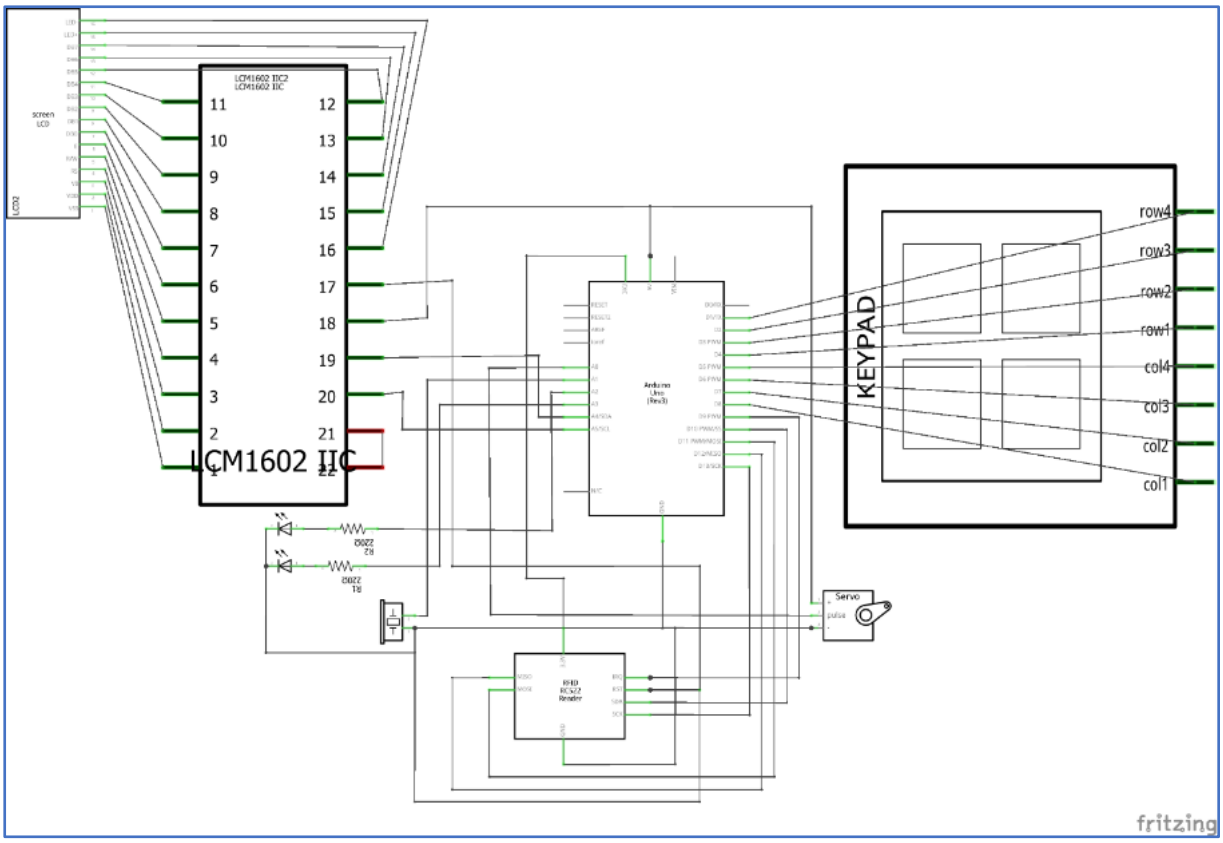


Figure 3-17 Schematic view of proposed solution.

3.4 Case design

After assembling the electronic parts, it was time to construct the exterior case, which would protect that interior mechanism. We chose the housing to be made of a carboard material to ease the prototyping process. First we had to disassemble the box and cut out necessary holes for the components' wiring, locking mechanism, Arduinos ports, reassemble it and see whether it will fit. We encountered a few obstacles in the form of having the steel lock fell of after applying only a minor force, faulty Chinese dupont wires, insufficient number of pins on the Arduino board and other minor hiccups. Therefore, we decided to carefully measure the length of our cables, dimensions of the components, so that we would be finally able to place them and the locking mechanism accordingly.

We succeeded on our third try. The first two times we made a few shortcuts, which resulted in components not fitting as expected or falling of the housing wall. Figure 3-18 depicts the effort to measure everything and reinforcing the box with additional cardboard pieces (from previous attempts) seen on the picture.

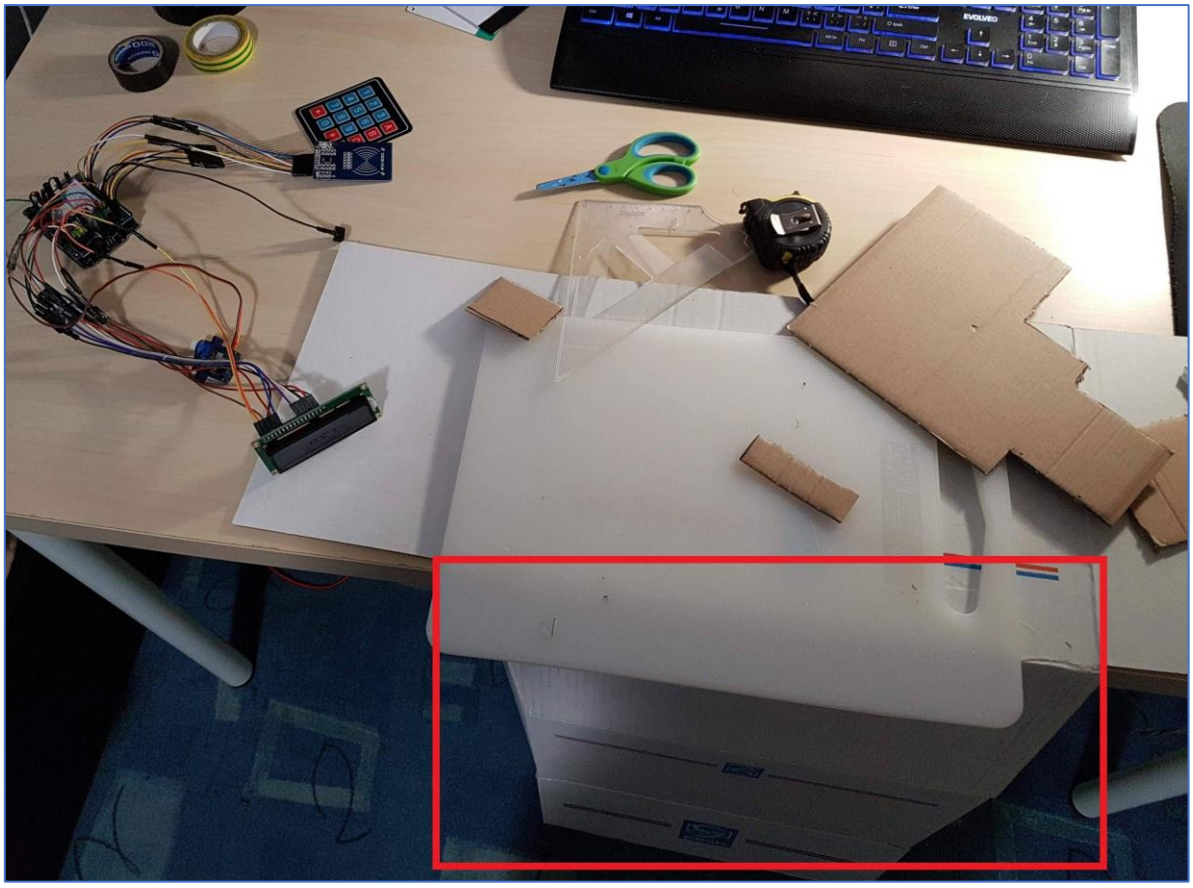


Figure 3-18 Prototyping the safes housing (1).

Figure 3-19 depicts the first design blueprint. On the right side, there is a custom unaltered cardboard in 2D. On the left side we can see the same carboard with the positions of the components indicated by a black marker. The positions were chosen based on the dimensions of the HW and the length of the cables available to us.

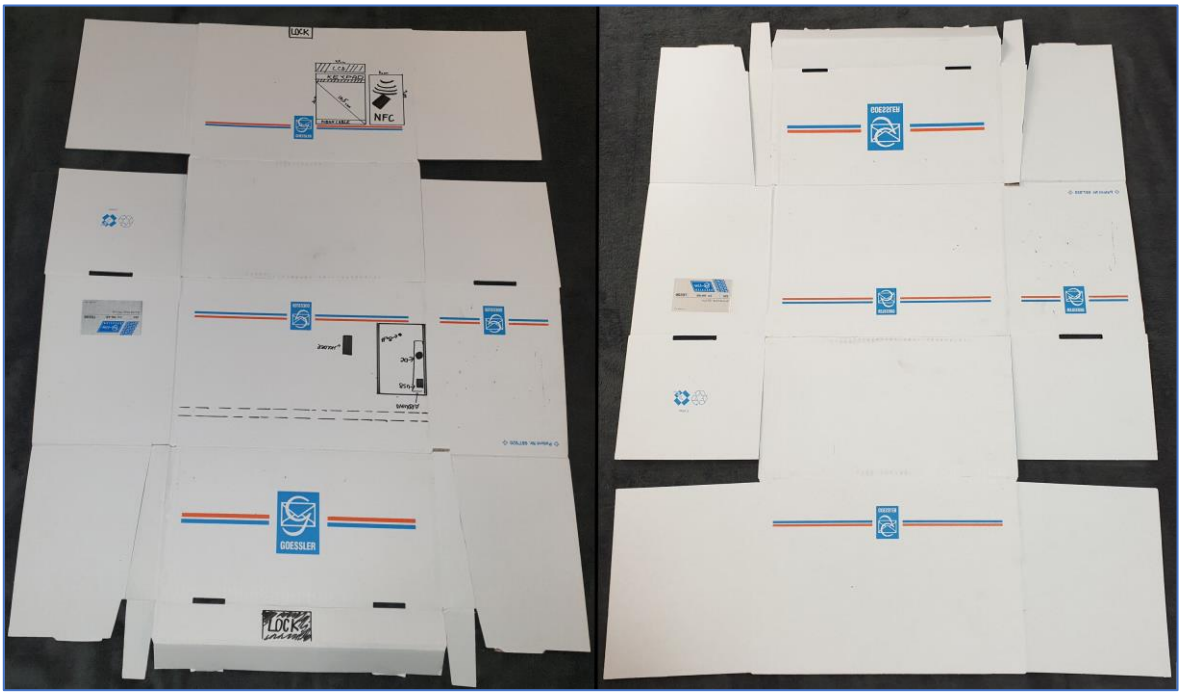


Figure 3-19 Prototyping the safes housing (2).

Figure 3-20 depicts the first successful iteration of our housing design without the LEDs, buzzer and having the first locking mechanism iteration already in place. At first, the locking mechanism was held in place only by duct tape but was later reinforced with bendable iron wires.

Figure 3-20 First iteration of the housing.

During the component measuring phase we also took into account the placement of the stickers made, cut and prepared in the FABLAB premises beforehand. Design of our logo was in a program called "*Inkscape*". We made a vector image in Inkscape and then "cleaned it up" in program called "*Graphtec Studio*" (ver. 2.2), so that the cut path for the vinyl cutter is unambiguous. The latter program was also used when "communicating" with the vinyl cutter. We used a vinyl cutter CE-6000-60 [3] from the Graphtec company available to public in the premises of "Science park UK" located in Bratislava at Ilkovičova 8. These custom-made vinyl stickers were then stored and put on the safe at a later time.

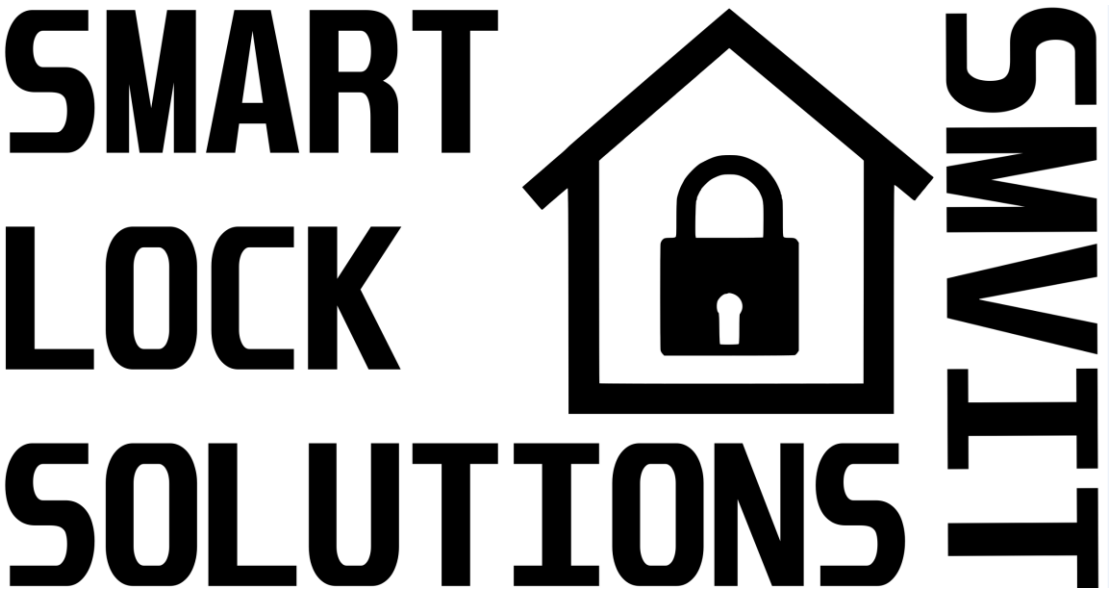


Figure 3-21 Vectorized design of our logo for the housing.

3.5 Verbal functionality description

We design and constructed a smart lock system to manage access to a physical location. This multi-input access control is achieved through authentication by PIN code (entered through a membrane keyboard) or by RFID tag compatible with the system. Cards compatible with the system can be found earlier in this work. User is informed about the outcomes of his/her actions through LCD screen, red/green LED diodes, active buzzer and a servo motor, which can lock/unlock the safe based on the validity of the given input. Functionality is covered later in this work and fully meets the criteria outlined in the requirements section at the beginning of this work. Functionality can be also deducted from the (well commented) source code or the bullet point functionality description later in this work.

Figure 3-22 depicts a (kind of) state diagram, which can be interpreted as a sequence diagram as well, starting in the locked state after the system powers up. It may seem incomplete (in regards to UML state diagram requirements), but its main purpose is to provide just the necessary information for an owner to be able to operate the safe after seeing this single diagram (guide). Several actions are omitted since they are easily deductible from the diagram. System can switch to states highlighted by red only when it is **locked**. It can switch to states highlighted by green color only when its **unlocked** and to states highlighted by turquoise when it is either **locked** or **unlocked**.

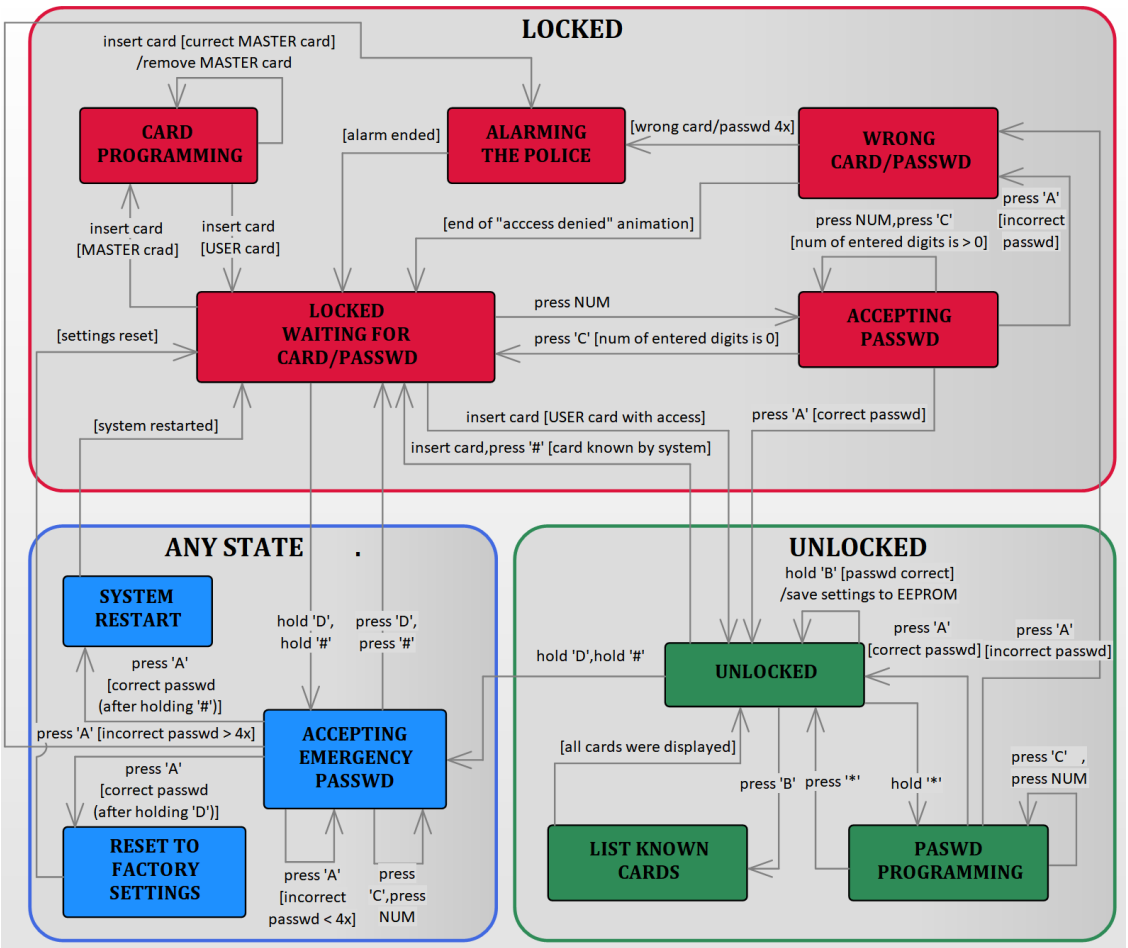


Figure 3-22 Pseudo-state diagram of the system.

Transitions in the previous state diagram are done based on the formula depicted on Figure 3-23. State transitions are initiated by the actions and their guards respectively (in the code by a concrete switch statements based on the pressed button). Every state has certain number of auditory/visual cues programmed into them. Every single LCD output can be found later in this work.

ACTION 1, ACTION 2 [GUARD FOR ACTION 1/2] / RESULT OF ACTION 1/2

Figure 3-23 State change formula.

During the boot, system loads all the saved RFID cards (henceforth only referred to as cards), loads main PIN from the long term memory (EEPROM) and switches to state "LOCKED WAITING FOR CARD/PASSWD" (henceforth only referred to only as LOCKED state) and locks itself. Generally, we can think of the system as either **locked** or **unlocked**. It can be unlocked either by entering a correct PIN code or inserting a known card. System recognizes and differentiates between 2 main types of cards: MASTER and USER card depending on their status during the saving process. It is possible to change a MASTER card into a USER card and USER card into a MASTER card. MASTER card can manage (add/remove) other cards (to/from) in the systems database (DB). After inserting the card, system switches to state "CARD PROGRAMMING". If the MASTER card is inserted again (meaning twice in a row), system removes it from the DB and requires the user to present other card, which will be used as a MASTER card. USER card is used only for locking/unlocking the system. If the system is in the state "CARD PROGRAMMING" and we insert any other non-MASTER card, it will be added (removed) to (from) the DB depending on whether it is already stored in the DB.

3.6 Bullet point functionality description

If the system is LOCKED:

- We can **unlock it by entering the PIN code consisting of numbers** (PIN can be changed). After entering the first digit the state is changed to "ACCEPTING PASSWD". For every digit we input, '*' will appear in the lower part of the display while entering the password.
    - Pressing 'C' deletes one digit.
        - If we press 'C', and all the digits from the lower part of the display are already deleted (meaning the lower part of LCD is empty), system returns to state "LOCKED".
    - Pressing 'A' makes the system verify the entered password.
        - If the password matches the one stored in the DB, it switches to state "UNLOCKED".
        - If the password doesn't match the one in the DB and number of unsuccessful attempts is less than 4, system switches to state "WRONG

CARD/PASSWD", conveys the information about an unsuccessful attempt, lowers the number of remaining tries before alarming the police and switches to state "LOCKED".

- If the password doesn't match the one in the DB and the number of unsuccessful attempts is 4 (or more based on current settings), system switches to state "WRONG CARD/PASSWD", conveys the information about an unsuccessful attempt, switches to state "ALARMING THE POLICE", remains locked and launches the alarm. After a short while it switches back to state "LOCKED" (for development purposes, otherwise it would be necessary to deactivate it of course).

- Pressing '#' immediately deletes all the (already entered) digits, locks the system and switches to state "LOCKED".

- We can **unlock** it by **inserting** a **USER card** registered in the systems database.
  - If the USER card **IS** registered in the DB, the information about successful attempt is shown and system switches to state "UNLOCKED".
  - If the USER card is inserted and it **IS NOT** registered in the DB, system evaluates it as a login attempt. If unsuccessful attempts number is less than 4, system switches to state "WRONG CARD/PASSWD", conveys the information about an unsuccessful attempt, lowers the number of remaining tries before alarming the police and stays in the "LOCKED" state.
- We can **add** or **remove cards** by inserting the MASTER card first. After inserting the MASTER card, system switches to state "CARD PROGRAMMING" and waits for a user to insert another card.
  - If the (next) inserted card is already in the system (and IS A MASTER), system removes it from the DB and waits for a user to insert another card which will function as a MASTER (since system CANNOT function without a MASTER card!).
    - If the inserted card is already in the DB as a USER card, it is removed as a USER card and instead registered as a new MASTER card.
    - If it is not in the system, the card is simply added as a MASTER to the DB.
  - If the inserted card is already in the DB as a USER card, it is removed from the DB.
  - If the inserted card is not already in the DB, it is simply added as a USER card.

<u>If the system is UNLOCKED:</u>

- We can **lock** it by inserting a USER card, MASTER card or by pressing '#'.
- We can display **unique identifiers (UUID) of the cards** stored in the DB. By pressing 'B' system switches to state "LIST KNOWN CARDS" and starts to list cards in

the DB (with MASTER card always being listed first).

- We can change the PIN code by holding '*' when system is unlocked. System switches to state "PASWD-PROGRAMMING" and shows a prompt for entering a current password before allowing a user to switch to a new password. During this part, system will behave identically to the state "ACCEPTING PASSWD".
    - With the difference that when all the digits are deleted (bottom part of LCD is blank) and user presses 'C' again, system will intentionally remain in this state.
    - Password change can be canceled by pressing '#'.

If the system is either LOCKED or UNLOCKED:

- We can **REBOOT** it by holding '#'. First we must verify our identity by entering a special emergency password (which is permanently set to "1154" for development purposes and can be changed only through modifying the code). After the confirmation of the emergency password, system informs the user about an upcoming system restart and initiates a special restart function (hard-wired by Arduinos engineers for emergency purposes – launching this function is identical to pressing the physical REBOOT button on the board).
- We can **RESET** the system to "**our factory settings**" by holding 'D'. First we need to enter a special emergency password (which is 1154 in this case as well), afterwards we are informed about the upcoming system RESET to factory settings. All the cards are deleted from the DB including the PIN code. User is informed that the PIN code is reset to "1234" and is forced to insert a card, which will be used as a MASTER from now on.
- We can **save current PIN and cards** (with their status) **to a long term memory** by holding 'B' (while the system is unlocked) if we want the changes to persist after a system restart (or power down). User is informed about the outcome of the attempt to save everything into the long-term memory.

3.7 Technical aspects

Name: Cute Buzzer Sounds Arduino Library (CuteBuzzerSounds.h)

Version: 1.0.0

Source: https://github.com/GypsyRobot/CuteBuzzerSounds

Description: Allows to make very interesting robotic sounds with the buzzer.

Name: EEPROM extension Library for Arduino (EPROMextent.h)

Version: 1.2.1

Source: https://github.com/Locoduino/EEPROMextent

Description: Allows to read/write complex data to/from the EEPROM of the device. It uses another library EEPROM.h to execute low level EEPROM access.

Name: Keypad library for Arduino by Mark Stanley, Alexander Brevig (Keypad.h)

Version: 3.1.1

Source: https://github.com/Chris--A/Keypad

Description: Enhanced keyboard library forked from Arduino.cc.

Name: LiquidCrystal Arduino library for the I2C LCD displays (LiquidCrystal_I2C.h)

Version: 1.1.3

Source: https://github.com/johnrickman/LiquidCrystal_I2C

Description: Allows to control an LCD screen connected to an I^2C module through an I^2C bus of the master device. Library uses a default I/O library called Wire.h for a low-level communication.

Name: Arduino RFID Library for MFRC522 (FRC522.h)

Version: 1.4.5

Source: https://github.com/miguelbalboa/rfid

Description: Allows to read/write from/to RFID cards with interface compatible with ISO/IEC 14443A Mifare. This library uses another library SPI.h to communicate between the device and the card reader through SPI interface.

Name: Servo Library for Arduino (Servo.h)

Version: 1.1.5

Source: Arduino IDE (libraries)

Description: Allows to control several different models of servo motors.

3.8 Code overview

Body of the main loop function is one big switch with states implemented through enum type to increase code readability and avoid string matching for such a basic purpose. States which can be activated are: LOCKED, UNLOCKED, PROGRAMMING_CARD, PROGRAMMING_PASSWD, DENIED, PASSWD, ALARM, OTHER, SYSTEM_RESETT. These states are described in detail earlier in this work (under a slightly different names in the diagram, but these are state names copied from the code).

Figure 3-24 depicts a code placed at the beginning of every state (of course with small variations depending on the states). We must check whether we came from another state and act accordingly. Main reason for these checks is to prevent the system from repeating the same actions, which are required to be executed only once when entering

a certain state (for example overwriting the text on the screen, constant signaling to lock something that is already locked, etc.).

```
case LOCKED:     /*System is locked, user has to authentify himself BY RFID card*/
  if (lastState != LOCKED) {
    //Serial.println("ENTERING STATE: " + String(state_name[state]));
    lastState = LOCKED;
    printLocked();
    cute.play(S_DISCONNECTION);
    LOCK();
    delay(500);
  }
```

Figure 3-24 Code preview of a section executed before entering a state.

Arduino **Uno** has a limited computing power and is a relatively simple device compared to a desktop computer. It cannot do a lot of things including a non-blocking/asynchronous operations by itself. Not even with the Scheduler.h library, which supports only Arduino DUE and more advanced devices. We quickly realized the need to do asynchronous tasks or at least some kind of mimicry of such behavior. After an exhausting amount of iterations we managed to solve it by starting a timer at the boot of the Arduino, measuring the elapsed time and executing certain behavior upon reaching a certain time intervals for every single operation needed to be executed asynchronously. It is a huge disadvantage when using a system like this. Figure 3-25 depicts a small part of such a non-blocking code to control a buzzer and LEDs in a certain manner.

```
/*Async blinking and buzzing func.*/
void programmingModeSigns(bool buzzerOn) {
  if (millis() - programming_led_timer > 100) {
    digitalWrite(R_LED_PIN, !digitalRead(R_LED_PIN));
    digitalWrite(G_LED_PIN, !digitalRead(G_LED_PIN));
    programming_led_timer = millis();
  }
  if (buzzerOn) {
    if (millis() - programming_buzz_timer > 1000) {
      tone(BUZZER_PIN, 1500, 100);
      programming_buzz_timer = millis();
    }
  }
}
```

Figure 3-25 Preview of our non-blocking code used to control a peripheral devices.

In the event of crossing the maximum number of allowed attempts to unlock the safe, system is simulating the sound of an alarm. System does not yet contain HW necessary to alert the police and contact the owner, but such an option could be added later (after switching to a device with more input pins). Multiple things are (mainly due to Arduinos memory limitations) solved through global variables, which are accessed in

a well-labeled and transparent way. Global variables are used only when necessary and are strictly marked by an underscore and a different naming convention.

At some point, we ran out of memory for variables and sketch (code) and we were in need of a serious code optimization/overhaul. At the preview presented earlier in this work, we can see that this optimized version uses 60% of maximum program storage space and 72% of dynamic memory even after a few iterations to eliminate every possible redundancy by using custom made print functions (to reduce sketch size), variable reuse, etc. When observing alternative solutions available on the internet before picking our processing unit, we thought that the Arduinos memory will be sufficient, but we didn't count on wanting to make the system so complex and elaborate including the effort to fix all extreme use cases.

3.9 List of system messages

This section contains all the messages, which we can encounter during the usage of the safe. These are various outputs informing us about the current system state and/or an action required. Messages are quite self-explanatory and were chosen in accordance with the LCDs limitations.


MASTER CARD LOST
SCAN NEW MASTER!
PROGRAM MODE
ADD/REMOVE CARDS
---CARD ADDED---
--CARD REMOVED--
ENTER OLD PASSWD
***
ENETER NEW PASS
AND PRESS 'A'
NEW PASSWORD:
****
NEW PASSWD IS:
1237890
SETTINGS SAVED
TO EEPROM :)
DATA LOADED
FROM EEPROM :)
CARDS SAVED: 4
1:C73F5062
3:9958A18B
MASTER CARD SET

4 CONCLUSION AND EVALUATION

Result of this semesters work is a successfully implemented prototype of an access control system (implemented as a safe) and an underlying documentation describing every aspect of our solution. It took approximately 220 to 230 man-hours to complete this project including the planning and all the failed attempts due to various reasons discussed throughout this work.

The safe is still only a prototype and would require several improvements in order to be a usable product. In the current state, all of the systems HW is vulnerable when the safe is opened, as well as the Arduinos USB port and the power supply cable, which can be accessed by anybody with a physical access to a locked device. However, this state is only temporary and is present to make the prototyping and development of the system easier.

Overall, we can conclude that the solution is currently in a very desirable state and covers all the intended functionality (and more), which was overhauled several times during the making. We grossly underestimated the time, hardware and knowledge required to complete this assignment, but in the end exceeded our own expectations of a resulting artifact. Difficulty grew disproportionally with every added device, due to reasons mentioned in the section describing the encountered difficulties.

4.1 Development experience

Making of the system was accompanied by quite a few HW and SW complications. We want to elaborate on this in more detail since there might be others who will find it useful before deciding to undertake such a task. First off, we need to state that we have used a soldering iron only a few times before this task. We had little to no prior knowledge about voltages, current, resistors, cable wiring, etc. It is not to say that those things are overly difficult, only that they have a specific learning curve and when encountering them for a first time without a guidance, it will take some time to grasp the basics (since one cannot know if the information on the internet is correct). We encountered people who couldn't grasp the code complexity of our solution, but had no problem in designing an Arduino circuit with multiple peripherals on the same PIN without damaging the devices (and similar obstacles). It only stands to show the importance of a prior experience and knowledge in any area.

Figure 4-1 depicts the preparation and analysis stage. (C) shows a resistor color code table, which we encountered for a first time in our lives and had to study it extensively, just to resort to a various youtube videos since the coloring proved to be totally unreliable and information on the sheet ambiguous. On the left side of the section (D) there is a first version of the code containing a core logic for our system on the paper. Right side contains research notes about the electronics (E), Arduino basics, wiring and similar topics needed for this project even before connecting Arduino for the first time. Later on, we had to study extensively how to work with the RFID equipment, cards, their

memory layout, the possibility of cloning it (rewriting its UUID) and similar topics depicted on Figure 4-3 section (B). The information about being able to overwrite a cards UUID proved to be incorrect (at least in our case).

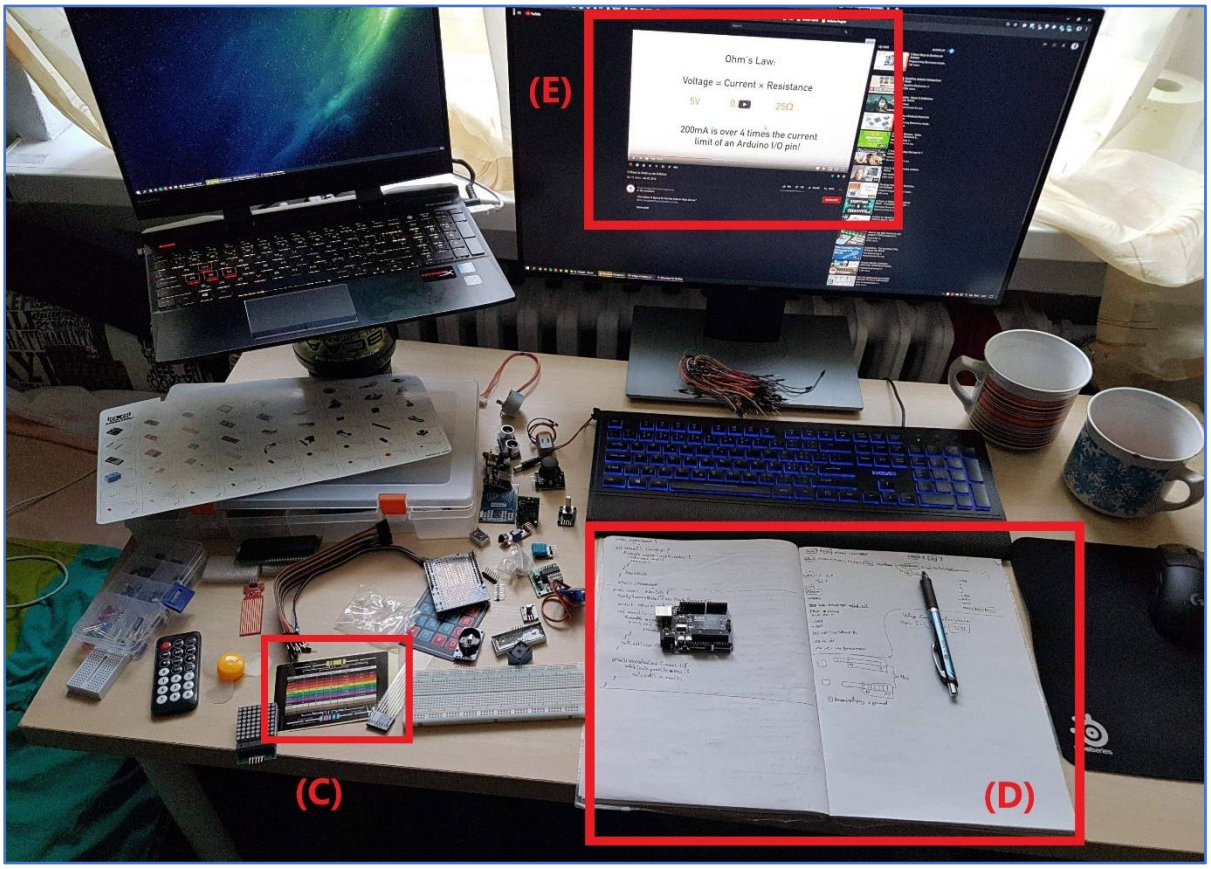


Figure 4-1 Workspace during development.

When dealing with a software (on a similar scale), it is quite easy to correct the mistake, but soldering/connecting something in the wrong way might result in a permanent damage (as we learned ourselves). Figure 4-3 section (A) depicts our soldering equipment, which proved to be extremely difficult to work with considering we had to solder pins smaller than the tip of the soldering iron and had almost zero experience beforehand. Since the devices came without pins soldered to it, we had to do it ourselves and Figure 4-2 depicts only one example of such an amateur soldering attempt.

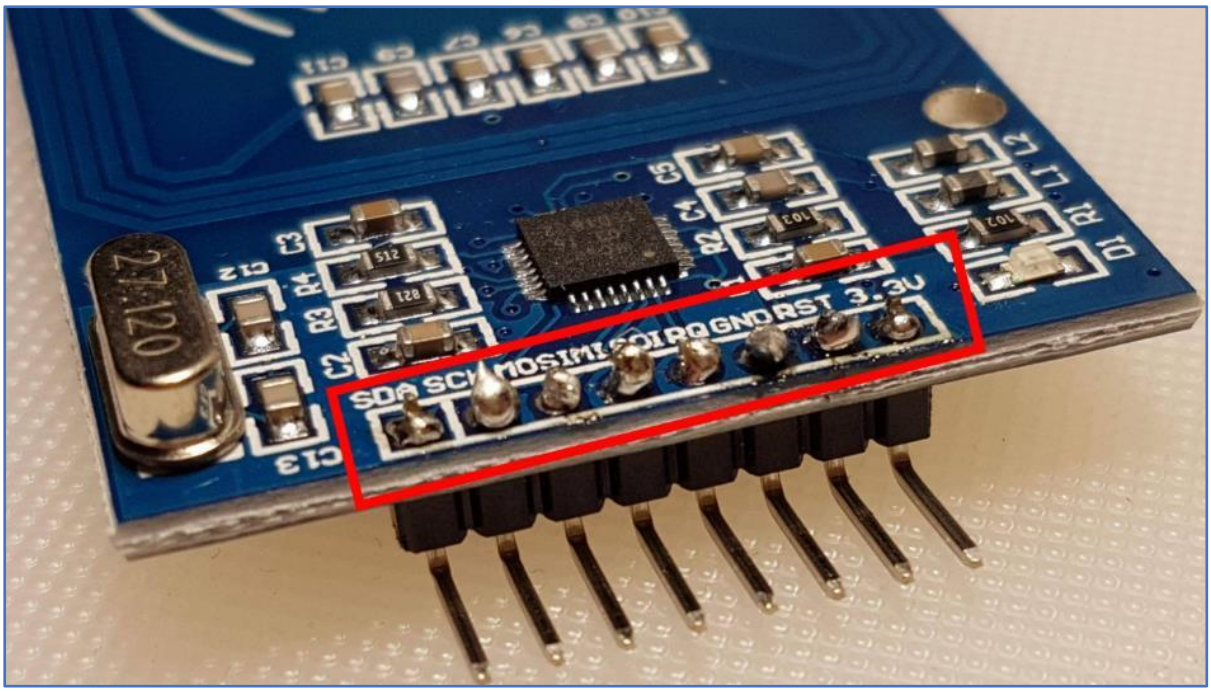


Figure 4-2 First time soldering attempt.

All of the dupont wires were of very low quality and a considerable number of those wires had a tendency to get loose easily or break altogether making us wonder why the code is not working since the device was detected, but was missing data from a certain PIN or two. It resulted in the need of repeated checking whether the problem is in the soldering, the dupont wires, the port on the Arduino, mis-declaration of a pin in the code, etc. Dupont wires bought from china were made quite cheaply and they were a source of many frustrating "incidents" since some of them worked literally just under a certain angle when inserted into female pin connectors on the board. Sometimes they broke after a few re-plugs (but were visually intact).

Another big endeavor was to unsolder 16 pins from the LCD screen without a de-soldering pump so we could later solder the LCD to the I^2C converter. It required us to repeatedly heat up several pins at the same time to able to move those few the pins only a quarter of a millimeter (since the solder stuck to the pins very well and was quite problematic to remove). After every attempt to modify HW, there was a need to retest the HW through code to make sure, that the soldering was successful and every wire was connected to the right female pin. When considering the quality of the dupont wires one can only imagine the patience needed for such a task.

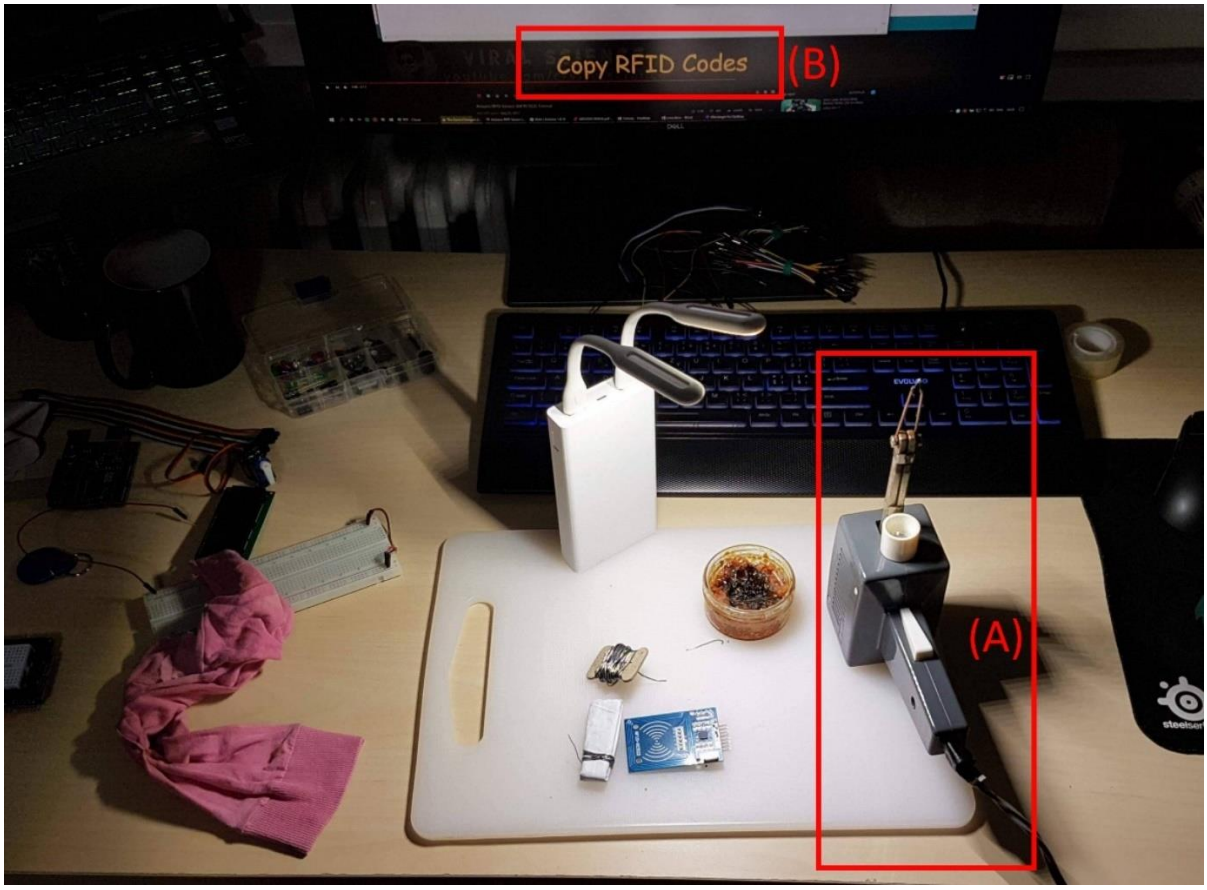


Figure 4-3 Used soldering iron example.

Making the hardware work was only half the battle since we were quickly met with development environment problems ranging from totally unusable IDEs (couldn't even create or be given a *.cpp file – turns out it was a known bug) to unexplainable and unexpected behavior in another IDE just to find out that it was a bug (as well) present in the fundamental String.h library used by the Arduino IDE. Arduino IDE has a set of core C and C++ libraries (as well as some of "its own implementations" managed by Arduino.cc and replacing the native C libraries) delivered with the IDE during its installation. Since we were working with strings quite a lot, we naturally used the libraries provided with the IDE and did not question its implementation. Some Arduino IDE custom string manipulation functions have unpredictable results, namely the function "*compareTo()*" and one other which sometimes work as they should and other times not quite as well. After exhaustingly checking for errors in our code we found out it is in fact a bug and was tested and reported by several programmers a long time ago (on "forum.arduino.cc"), but was not repaired to this date. Therefore we resorted to using only native C functions + libraries made for the peripherals.

4.2 Encountered difficulties

There were some cases of integer overflow because of the Arduinos architecture limitations and a way we use our async timer. We needed to change the type of a timer variables to be able to contain a very huge number since we are storing "milliseconds passed since the Arduino board began running the current program". This number will overflow (go back to zero) after approximately 50 days according to a documentation. Since returning Integer is of type "unsigned long" (~32 bits), we resorted to using type

“uint64_t” (~64 bits) for the timer, which should be more than enough to handle such a number, but might be cause for some unexpected behavior in the future (probably in some extreme cases). This technical aspect is mentioned in this section because we came to a conclusion that even functions guaranteed to work are not to be relied upon.

Working with EEPROM and modifying it proved to be a challenge as well (probably due to a variation in existing devices the library is intended for). Either the library EEPROM.h or the way Arduino is working with EEPROM caused some unexpected behavior during its manipulation. Even though we made sure we are reading/writing EEPROM in the correct positions, the only “solution” was (and still is) to initialize first 512 bytes of Arduinos EEPROM to ‘0’s every time we want to save new system settings. That will result in an increased EEPROM wear since it is an *electrically erasable programmable read-only memory* and is designed to handle ~100,000 read/erase cycles. This might pose a problem during a long-term usage when the memory becomes so wore down that future manipulation of its contents will not be possible and again may lead to some unexpected behavior.

Problems listed were only a subset of struggles encountered during the construction of our prototype. In several different points in time we were in need of additional equipment because the available parts weren’t functioning and they needed to be replaced. This resulted in a lot of wasted time since we needed to find, order, wait and retrieve the required parts from several different stores in a acceptable time frame (depending on their availability and price), which in turn led to a partial halt in the development process.

4.3 Lessons learned and hindsight

During the work on this assignment we undoubtedly grew our knowledge base and skill set by a significant amount in these fields, in which we had little to no prior experience (except for programming). We learned basics of circuit engineering, tried working in several IDEs including the fritzing (CAD software for circuit design), learned how to work with a vinyl cutter, practiced our soldering skills and problem-solving abilities, tested our imagination with respect to constructing a visually pleasing prototype and without a doubt improved our patience. It is hard to say what would we have done differently in hindsight since we still do not deem it viable to buy special equipment just for a one-time use.

In any case, have we had to give ourselves some advice before starting the work on this project, it would be the following:

- Buy a high quality dupont wires and certainly longer ones.
- Buy a de-solder pump and a good soldering iron
- Buy a more powerful computing device with a more advanced instruction set (as to avoid unnecessary problems with outdated library functions (there is no need to reinvent the wheel) and have more PINs to spare.
- Use the IDE recommended (and used) by the majority of casual user. In our case it was Arduino IDE.
- Have a person with prior experience in the field available for consultations.

- Check the exact state of PINs before buying peripherals (soldered or just disclosed).
- Be skeptical of using any provided library function.
- Really only assemble the device once it is guaranteed that it will not be needing another modification.

Apart from the previously mentioned hurdles, big part of the course related to this project was working with software called "Enterprise architect" (from now on EA). It was not mention earlier in the analysis part since one of the two main goals of this course was to gain experience with EA and it was not optional. EA is a visual platform for designing and constructing software systems for business process modeling and for more generalized modeling purposes based on OMG UML. Throughout this course and while working on this project we learned how to work with EA, namely how to store various aspects of our work in the data model, logically structure the information, export it to a common format (HTML and PDF) and publish it online for others.

4.4 Future work

One of the main drawbacks is the absence of the GSM and Wi-Fi module, which would allow the owner to lock/unlock the system from virtually anywhere. Another possible improvement would be to hide exposed parts and internal wiring such as it would not be even visible. Adding an ability to log accesses and the card/PIN which unlocked the system would be a great addition as well as the ability to approve access to system remotely (not to be confused with remotely unlocking the system). If we do not take into account the material our prototype is made of, next point of failure would be the servo motor, which can be easily broken and the safe pried open. This could be mitigated with the usage of electromagnetic lock (welded to the safe from inside) widely used as means of controlling access to buildings. Last possible improvement could be made by swapping the processing unit along with the board in order for it be able to accommodate additional pins needed for further functionality expansion.

4.5 Photographic documentation

This section contains more photographic artifacts captured during the work on this project. It captures our prototype in various different stages of its visual and functional development. We can clearly the small incremental improvements in terms of design and used HW throughout the semester. For example, the solution was never intended to have LED diodes, buzzer, its own power supply, I2C converter (such improvements were made to the code base in a similar fashion).

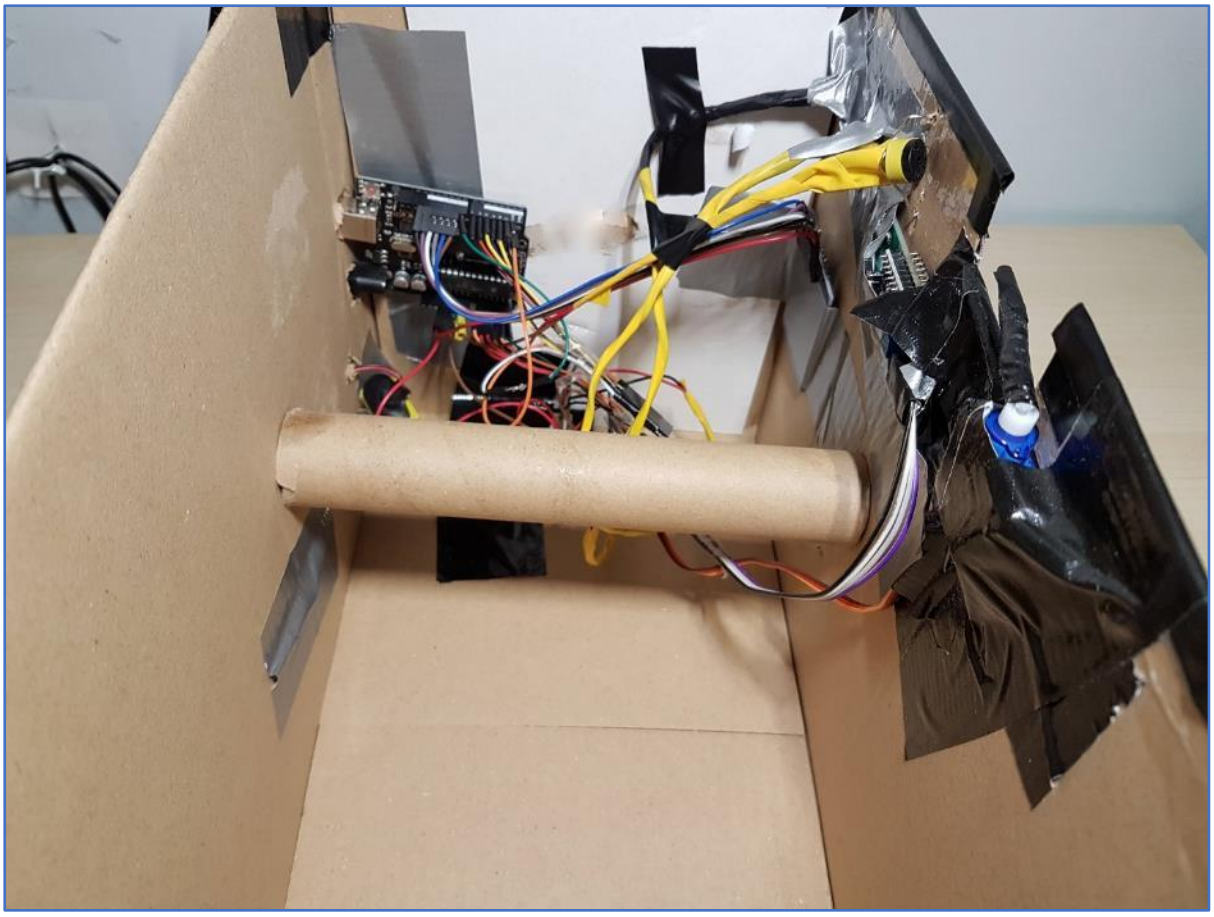
4.5.1 Earlier stages

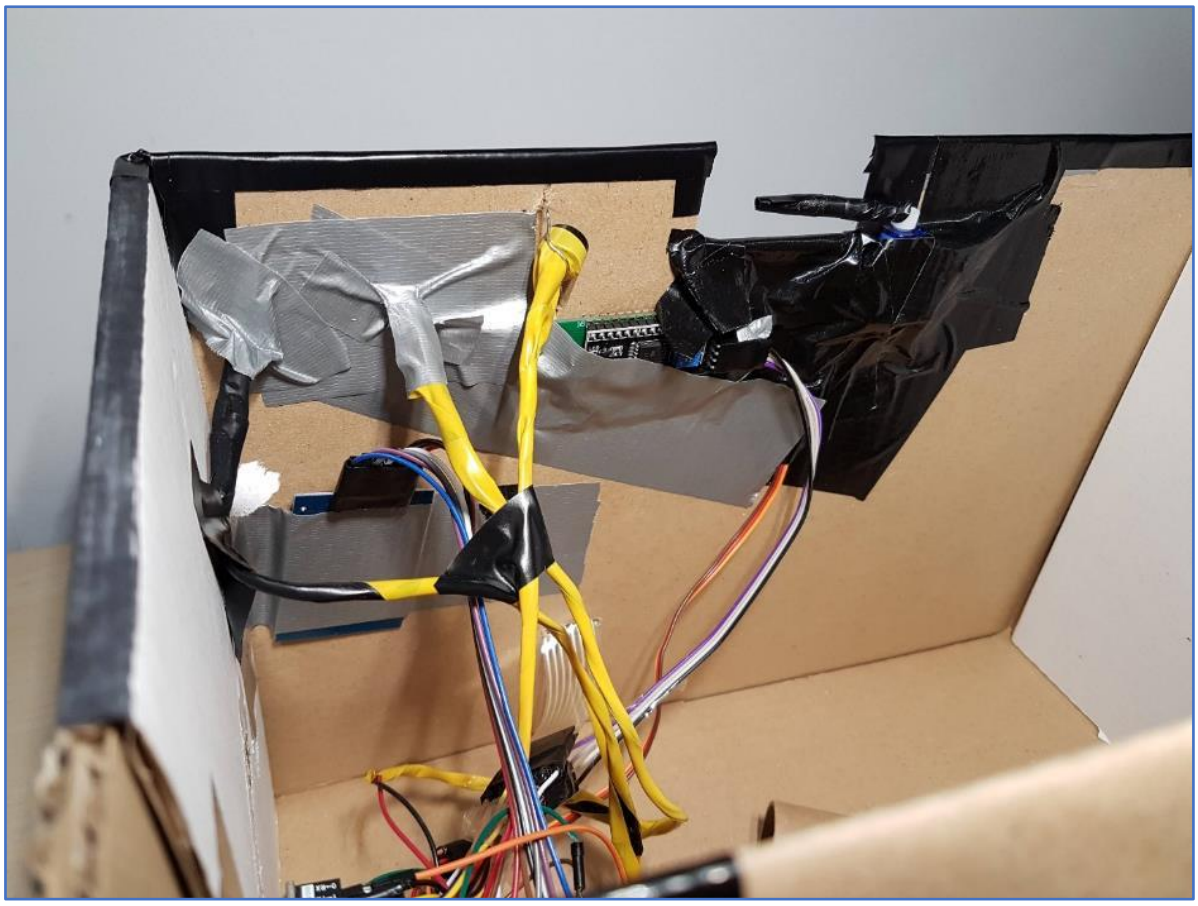
This section contains prototype photos from various stages of the project.

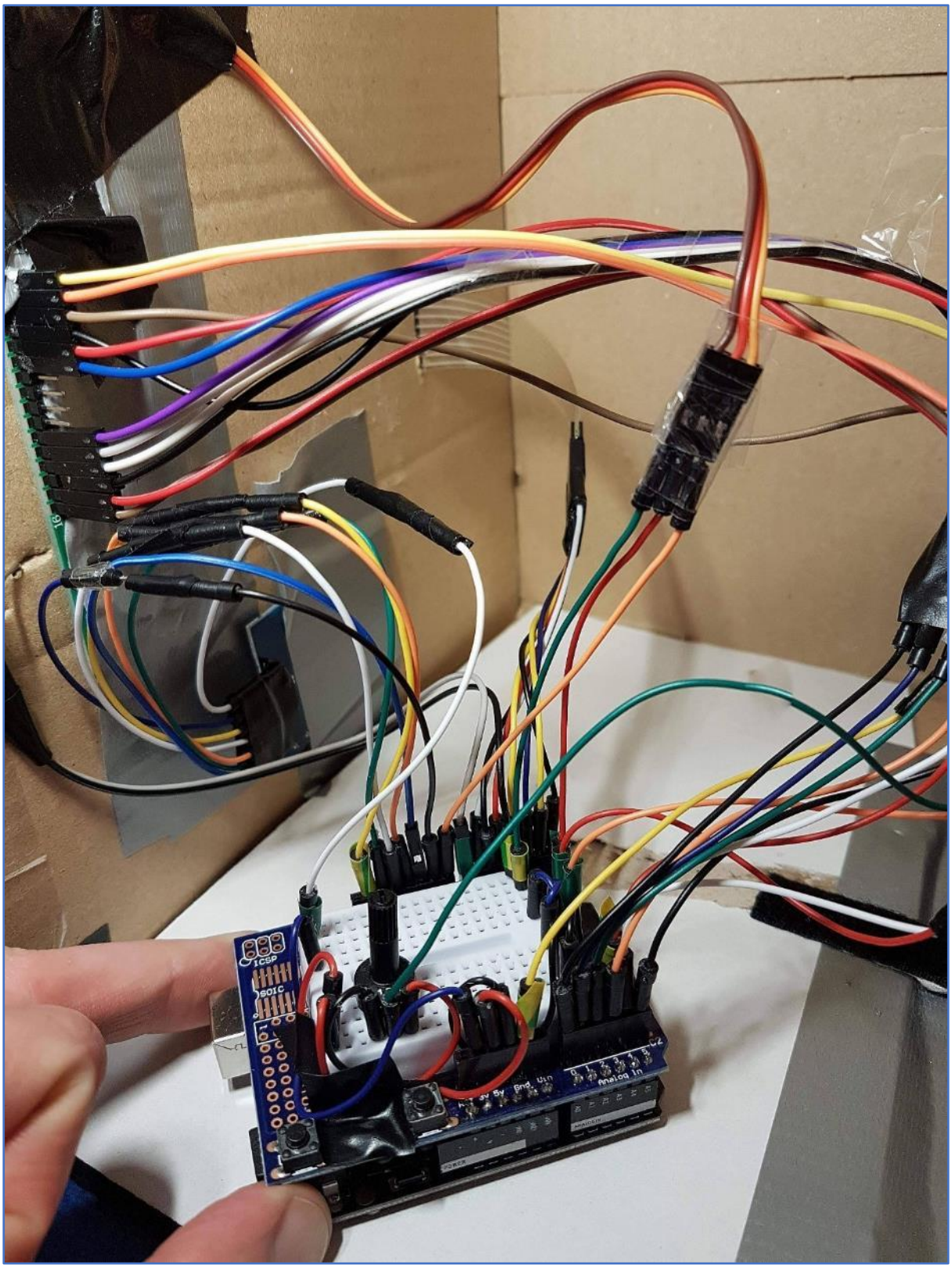

GOESSLER
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
NFC
GOESSLER


GOESSLER
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
NFC


GOESSLER

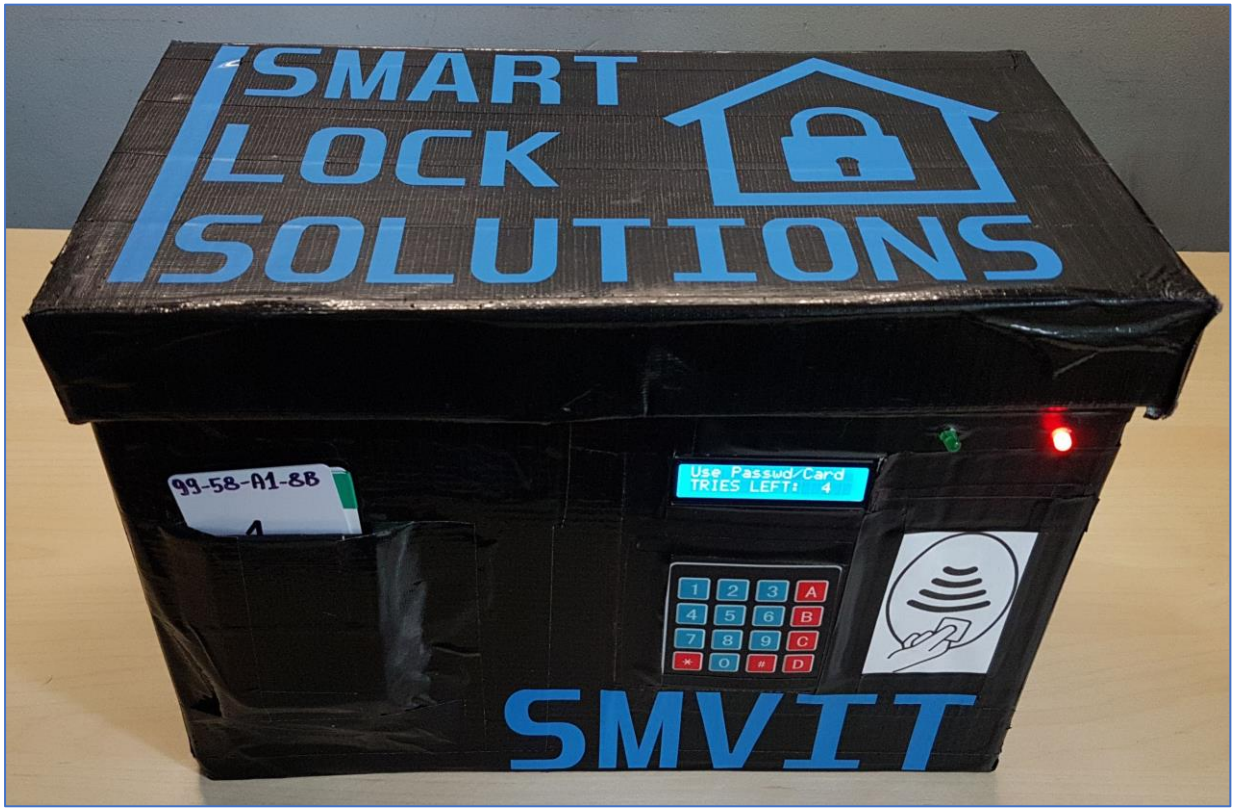


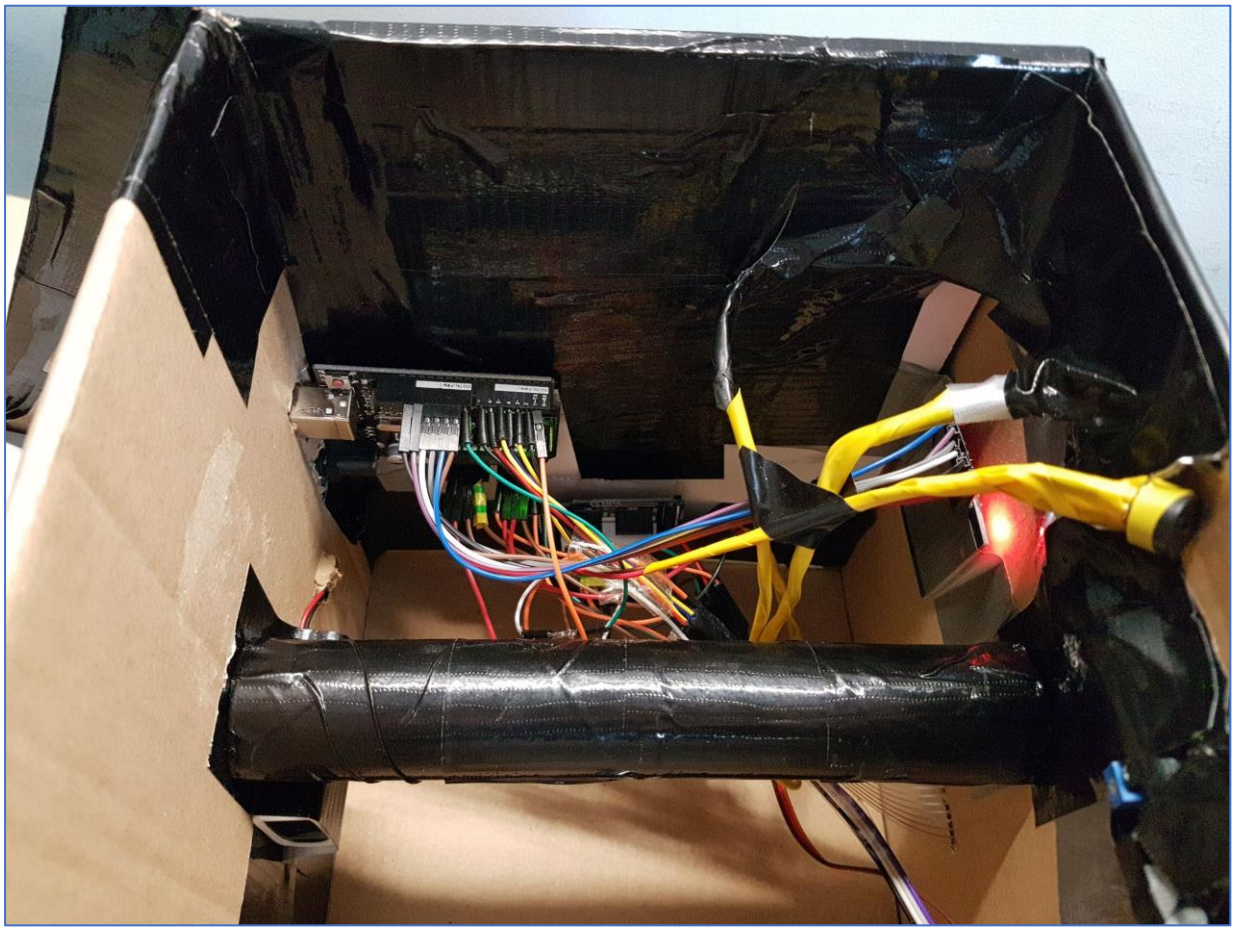


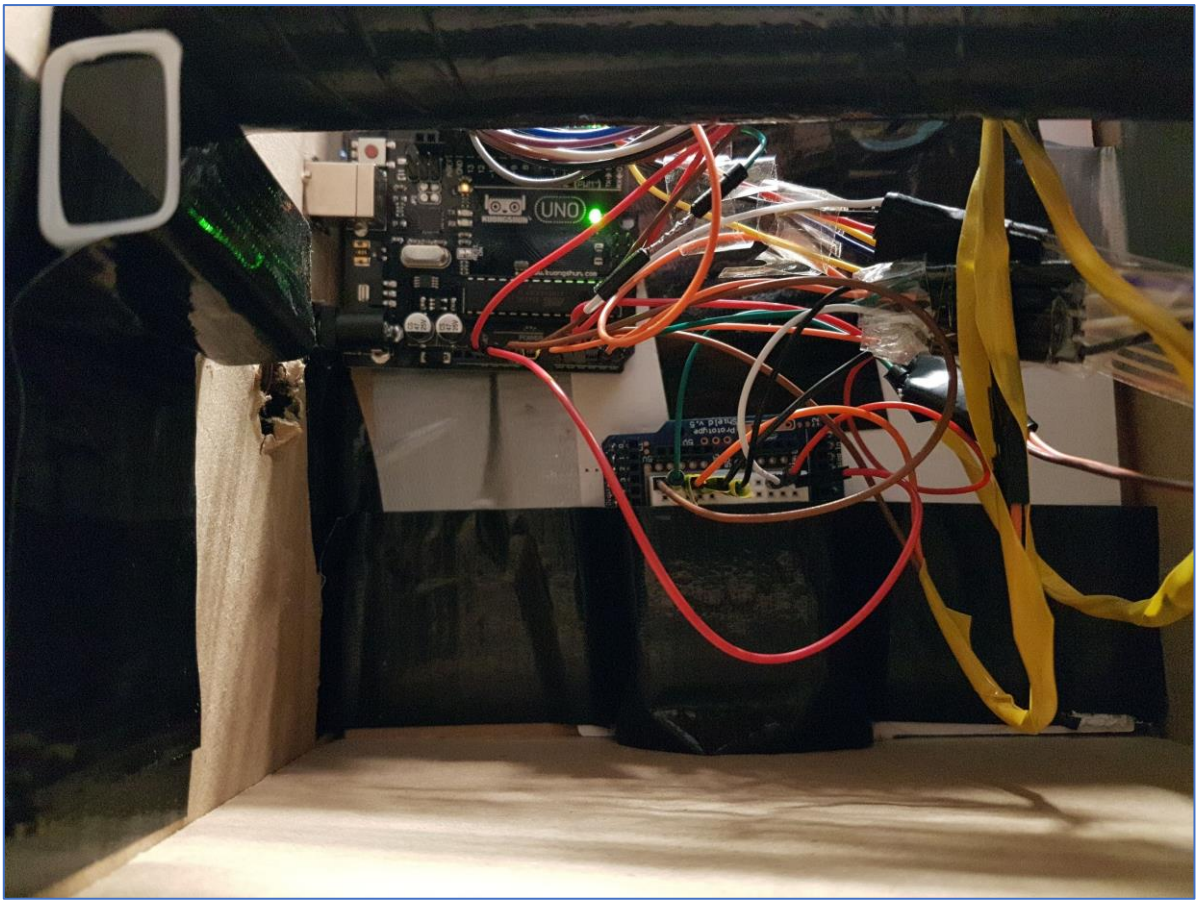

ICSP
Analog In

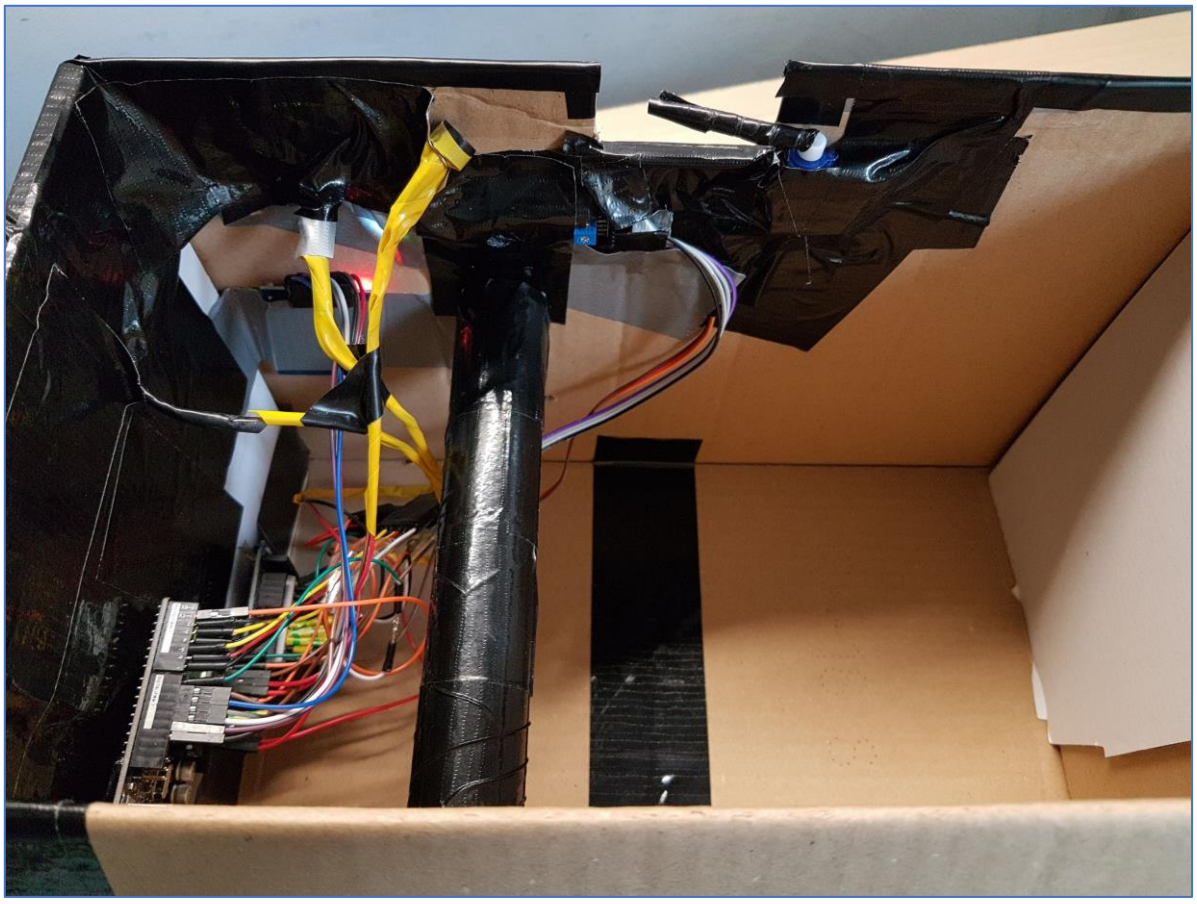
4.5.2 Final stage

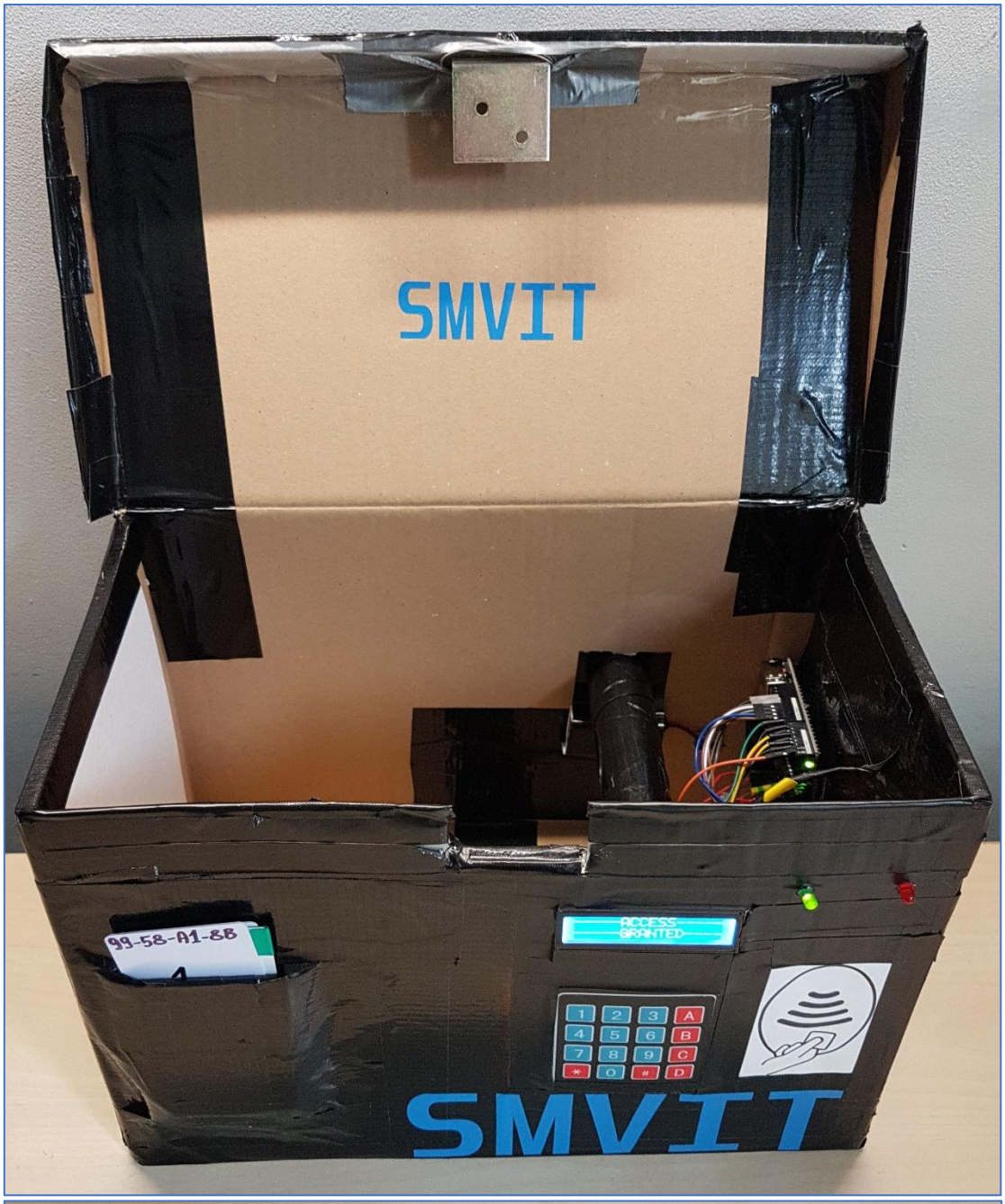
This section contains prototype photos in its final stage.

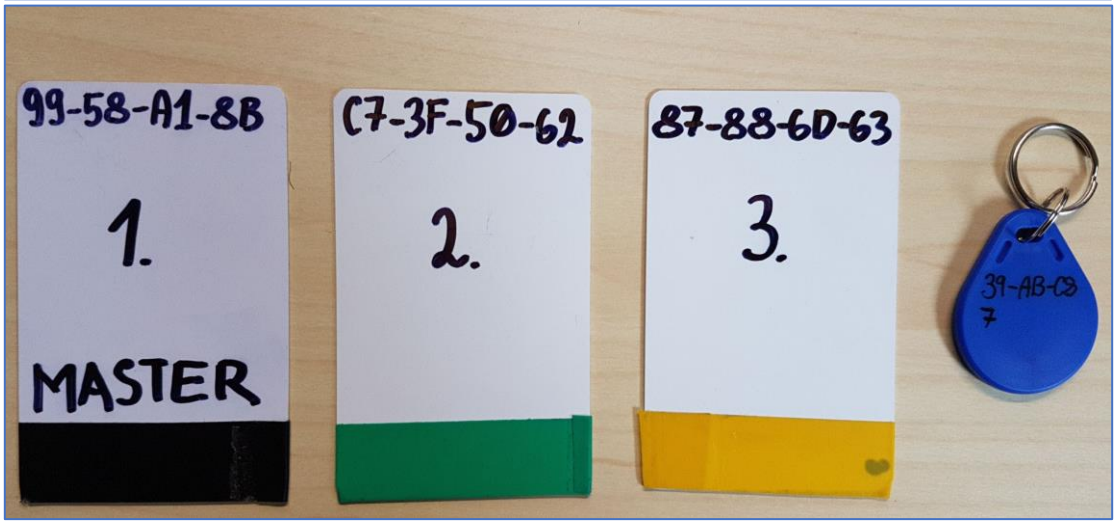





UNO




SMVIT
99-58-A1-8B
ACCESS GRANTED
SMVIT


99-58-A1-8B
1.
MASTER
C7-3F-50-62
2.
87-88-6D-63
3.
39-AB-C8
7

5 APPENDIX A – SOURCE CODE

```
#include <CuteBuzzerSounds.h>
#include <EEPROMextent.h>
#include <Keypad.h>
#include <LiquidCrystal_I2C.h>
#include <MFRC522.h>
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
/*****************   PIN ALLOCATION   *******************/
#define SERVO_PIN    A0
#define BUZZER_PIN   A1
#define G_LED_PIN    A2
#define R_LED_PIN    A3
#define RFID_RST_PIN 9
#define RFID_SS_PIN  10  /*SDA*/

const byte KP_ROWS       = 4;
const byte KP_COLS       = 4;
byte KP_rowPins[KP_ROWS] = {8, 7, 6, 5};
byte KP_colPins[KP_COLS] = {4, 3, 2, 1};
char hexaKeys[KP_ROWS][KP_COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

Keypad kp = Keypad(makeKeymap(hexaKeys), KP_rowPins, KP_colPins, KP_ROWS, KP_COLS); /*Init of 4x4 keypad*/
LiquidCrystal_I2C lcd(0x27, 20, 2);        /*Definition of LiquidCrtrystal LCD on 0x27 (size 16x2)*/
MFRC522 mfrc522(RFID_SS_PIN, RFID_RST_PIN);/*Definition of MFRC522*/
Servo myservo;                             /*Definition of SG-90 servo*/

const uint8_t PASSWD_MAX_LEN = 16; /*STORED password MAX length*/
const uint8_t MAX_ATTEMPTS = 3;    /*MAX authentication attempts before launching an alarm*/
const uint8_t CARD_ARR_LEN = 6;    /*MAX number of stored cards*/
const uint8_t UUID_LENGTH = 9;     /*UUID string length (8) + (1) '\0'*/

char password_stored[PASSWD_MAX_LEN] = {}; /*passwd store in EEPROM*/
char password_emergency[] = {'1', '1', '5', '4', '\0'}; /*emergency passwd for RESTART a FACTORY RESET*/
char password_tmp[PASSWD_MAX_LEN];                     /*TMP passwd variable when typing*/
int8_t passwd_tmp_digit_count = 0;                    /*TMP passwd counter to know position and length*/

bool passwd_change_flag = false;             /*FLAG to enter passwd change mode*/
bool arduino_resart_flag = false;            /*FLAG to RESTART system*/
bool pressed_B_button = false;               /*FLAG to check if B button was held or pressed */
char key_pressed = 0;                        /*TMP var to store currently pressed BUTTON*/

char *known_cards[CARD_ARR_LEN] = {};        /*STORED CARDS, [0] is MASTER card*/
int8_t known_card_count = 0;                 /*stored cards counter*/
int8_t card_pos = -1;                        /*VAR to store act card [0 to PASSWD_MAX_LEN] if in system,
                                               -1 if unknown*/
String act_card_id = "";                     /*global UUID of ACT SCANNED CARD*/

int8_t failed_attempts = 0;                  /*failed attempts to unlock system*/

uint64_t programming_led_timer;             /*TIMER for ASYNC LED blinking*/
uint64_t programming_buzz_timer;            /*TIMER for ASYNC BUZZER buzzing*/

enum STATE_TYPE                            /*STATES systeam can be IN*/
{
  LOCKED,
  UNLOCKED,
  PROGRAMMING_CARD,
  PROGRAMMING_PASSWD,
  DENIED,
  PASSWD_ENTERING,
  ALARM,
  OTHER,
  SYSTEM_RESETT
};
/*For DEBUG to print system state since enum is type INT*/
const char *state_name[] = {"LOCKED", "UNLOCKED", "PROGRAMMING_CARD", "PROGRAMMING_PASSWD", "DENIED",
                            "PASSWD_ENTERING", "ALARM", "OTHER","SYSTEM_RESETT"};
/*States are primarly stored so some task are launched only once entering the state*/
STATE_TYPE state;           /*Current system state*/
STATE_TYPE lastState;       /*Previous system state*/
/*****************************************************************************/
/*SOFT-RESTART arduino function*/
void (*resetFunc)(void) = 0;
```

```
/*Clears display and prints text to both rows based on length*/
void pprint(String text) {
  if (text.length() < 17) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(text);
  } else {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(text.substring(0, 16));
    lcd.setCursor(0, 1);
    lcd.print(text.substring(16));
  }
}

/*Prints and clears '*' during passwd typing*/
void lcdBottomPrint(int8_t pos, char toPrint) {
  if (toPrint == '*') {
    lcd.setCursor(pos, 1);
    lcd.print(toPrint);
  } else if (toPrint == ' ') {
    lcd.setCursor(pos, 1);
    lcd.print(toPrint);
  } else if (toPrint == '#') {
    lcd.setCursor(0, 1);
    lcd.print("                ");
  } else if (isdigit(toPrint)) {
    lcd.setCursor(pos, 1);
    lcd.print(toPrint);
  }
}

void printLocked() {
  //pprint("System is LockedUse Passwd/Card");
  pprint("Use Passwd/Card TRIES LEFT:  " + String(MAX_ATTEMPTS - failed_attempts + 1));
}

/*Prints UUIDs of cards currently in system*/
void printKnownCards() {
  pprint("CARDS SAVED: " + (String) known_card_count);
  delay(750);

  for (int8_t i = 0; i < known_card_count; i++) {
    pprint((String)(i + 1) + ":" + known_cards[i]);
    delay(3000);
  }
  lastState = OTHER;
  state = LOCKED;
  printLocked();
}

void LOCK() {
  myservo.write(3);
  state = LOCKED;
  digitalWrite(R_LED_PIN, HIGH);
  digitalWrite(G_LED_PIN, LOW);
}

void UNLOCK() {
  myservo.write(180);
  state = UNLOCKED;
  digitalWrite(R_LED_PIN, LOW);
  digitalWrite(G_LED_PIN, HIGH);
}

/*Func for card reading.Stores the card in global VAR 'act_card_id'.Upon SUCCESS read, returns true.*/
bool scanCardID(){

  if (mfrc522.PICC_IsNewCardPresent() == false) {   /*Card detected*/
    return false;
  }
  if (mfrc522.PICC_ReadCardSerial() == false) {    /*Start reading the card*/
    return false;
  }
  act_card_id = "";
  for (uint8_t i = 0; i < 4; i++) {               /*Used MIFARRE cards, have 4-BYTE UUID*/
    act_card_id.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  act_card_id.toUpperCase();
  mfrc522.PICC_HaltA();                           /*Stop reading since we only use UUID*/
  return true;
}
```

```
/*Checks if current card is stored in system (cards are stored in global VAR known_cards).If yes, retu
  returns a position of the card in array.*/
int8_t isCardStored(String tag) {
  for (int8_t i = 0; i < CARD_ARR_LEN; i++) {
    if (tag.compareTo(known_cards[i]) == 0) {
      return i;
    }
  }
  return -1;
}

/*Copies the last card in the system to the position of the one being removed and then frees last pos.*/
void removeTag(int8_t tag) {
  int8_t end_of_array = (known_card_count - 1);
  strcpy(known_cards[tag], known_cards[end_of_array]);
  free(known_cards[end_of_array]);
  known_cards[end_of_array] = NULL;
  known_card_count--;
}

/*Async blinking and buzzing func.*/
void programmingModeSigns(bool buzzerOn) {
  if (millis() - programming_led_timer > 100) {
    digitalWrite(R_LED_PIN, !digitalRead(R_LED_PIN));
    digitalWrite(G_LED_PIN, !digitalRead(G_LED_PIN));
    programming_led_timer = millis();
  }
  if (buzzerOn) {
    if (millis() - programming_buzz_timer > 1000) {
      tone(BUZZER_PIN, 1500, 100);
      programming_buzz_timer = millis();
    }
  }
}

void soundSuccess() {
  cute.play(S_HAPPY);
}

void soundButton() {
  tone(BUZZER_PIN, 1500, 300);
}

void soundFail() {
  tone(BUZZER_PIN, 200, 500);
}
/* Alarm sound*/
void soundAlarm(int duration) {
  int64_t tmpStartTime = millis();
  int64_t tmpIntervalTimer = 0;

  while (millis() - tmpStartTime < duration) {
    if (millis() - tmpIntervalTimer > 1000) {
      tone(BUZZER_PIN, 300, 400);
      tmpIntervalTimer = millis();
    }
  }
}

/*Clears current password tmp global VAR and length counter*/
void resetPasswordTmpCounter() {
  memset(password_tmp, 0, PASSWD_MAX_LEN * sizeof(password_tmp[0]));
  passwd_tmp_digit_count = 0;
}

void resetFailedAttempts(){ /*Resets the counter for unsuccessful attempts to unlock the system*/
  failed_attempts = 0;
}

/*Checks if currently typed password is correct*/
bool isPasswordCorrect() {
  if (strncmp(password_tmp, password_stored, PASSWD_MAX_LEN) == 0) {
    resetPasswordTmpCounter();
    resetFailedAttempts();
    return true;
  }
  resetPasswordTmpCounter();
  return false;
}

/*Delets all stored cards from system IN CURRENT SESSION (NOT from EEPROM)*/
void removeAllCards() {
  for (int8_t i = 0; i < known_card_count; i++) {
    memset(known_cards[i], 0, UUID_LENGTH * sizeof(char));
```

```
  }
  known_card_count = 0;
};

/*Writes current DEFAULT PASSWD and CARDS to EEPROM*/
void writeToEEPROM() {
  int8_t tmpPasswdLen = strlen(password_stored);
  int8_t sizecount = 2;

  EEPROMextent.clear(0, 512);
  EEPROMextent.writeByte(0, known_card_count); /*Need to know card count for later reading from EEPROM*/
  EEPROMextent.writeByte(1, tmpPasswdLen);    /*NTK stored passwd length for later reading from EEPROM*/

  /*Stores cards from current session one-by-one to EEPROM*/
  for (int8_t i = 0; i < known_card_count; i++) {
    EEPROMextent.writeString(sizecount, known_cards[i]);
    sizecount += UUID_LENGTH;
  }
  /*Stores default_password which user may change during a session*/
  EEPROMextent.writeString(sizecount, password_stored);
  if(state != SYSTEM_RESETT){
    pprint(" SETTINGS SAVED   TO EEPROM :)");
    delay(2500);
  }
}

void loadFromEEPROM() {
  char tmpText[15] = {};
  int8_t sizecount = 2;
  int8_t tmpCardCount = EEPROMextent.readByte(0);
  int8_t tmpPasswdLen = EEPROMextent.readByte(1);

  removeAllCards();
  for (int8_t i = 0; i < tmpCardCount; i++) {
    EEPROMextent.readString(sizecount, & tmpText[0], UUID_LENGTH);
    known_cards[known_card_count] = strdup(String(tmpText).c_str());
    known_card_count++;
    sizecount += UUID_LENGTH;
  }
  EEPROMextent.readString(sizecount, & tmpText[0], tmpPasswdLen);
  tmpText[tmpPasswdLen] = '\0';
  strcpy(password_stored, String(tmpText).c_str());
  pprint("  DATA LOADED     FROM EEPROM :)");
  delay(1000);
}

/*Special system events, excluded bcs its easier to capture LONG-PRESS like this*/
void keypadEvent(KeypadEvent key) {

  switch (kp.getState()) {
  case HOLD:
    if (key == '#') {
      // Serial.println("FROM KEYPAD EVENT: " + String(key));
      arduino_resart_flag = true; /*Flag for just a RESTART used later in the switch*/
      lastState = OTHER;
      state = SYSTEM_RESETT; /*restart and reset states are merged, thats why I use a restart flag*/
      break;
    }
    if (key == '*') {
      // Serial.println("FROM KEYPAD EVENT: " + String(key));
      if (state == UNLOCKED) {
        lastState = state;
        state = PROGRAMMING_PASSWD; /*State to change default_passwd*/
      }
      break;
    }
    if (key == 'B') {
      // Serial.println("FROM KEYPAD EVENT: " + String(key));
      pressed_B_button = false;
      if (state == UNLOCKED) {
        writeToEEPROM();
         lastState = OTHER;
        state = UNLOCKED;
      }
      break;
    }
    if (key == 'D') {
      lastState = OTHER;
      state = SYSTEM_RESETT; /*RESETS the system to "Factory defaults"*/
       break;
    }
    break;
  case PRESSED:
    if (key == '#') {
```

```
      // Serial.println("FROM KEYPAD EVENT RELEASED: " + String(key));
      if (state = UNLOCKED) {
        resetPasswordTmpCounter(); /*Locks the system if unlocked
        */
        state = LOCKED;
      }
      break;
    } else if (key == 'B') {
        pressed_B_button = true;
    }
    break;
  case RELEASED:
    if (key == 'B') {
//      soundSuccess();
//      delay(1500);
      // Serial.println("FROM KEYPAD EVENT RELEASED: " + String(key));
      if ((state == UNLOCKED) && (pressed_B_button == true)) {
        printKnownCards();
        delay(1000);
        lastState = OTHER;
        state = UNLOCKED;
      }
      break;
    }
    break;
  case IDLE:
    break;
  default:
    break;
  }
}

/***************       SETUP       **************/
void setup() {

  SPI.begin();
  cute.init(BUZZER_PIN);
  mfrc522.PCD_Init();
  myservo.attach(SERVO_PIN);

  lcd.init();
  lcd.begin(16, 1);
  lcd.backlight();

  kp.begin(makeKeymap(hexaKeys));
  kp.addEventListener(keypadEvent);
  kp.setHoldTime(2000);

  pinMode(R_LED_PIN, OUTPUT);
  pinMode(G_LED_PIN, OUTPUT);

  LOCK();
  memset(password_stored, 0, PASSWD_MAX_LEN * sizeof(char));
  char tmpPass1[] = {'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'};
  memcpy(password_stored, tmpPass1, sizeof(tmpPass1));
  loadFromEEPROM();
  lastState = OTHER;
  state = LOCKED;
  printLocked();

}

/*******************       LOOP       *****************/
void loop() {

  switch (state) {
    /***************       LOCKED       **************/
  case LOCKED:     /*System is locked, user has to authentify himself BY RFID card*/
    if (lastState != LOCKED) {
      //Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = LOCKED;
      printLocked();
//      pprint(password_stored);
//      delay(2000);
      cute.play(S_DISCONNECTION);
      LOCK();
      delay(500);
    }
    /*System is locked. After detecting keypress, enter 'passwd entering' STATE*/
    key_pressed = kp.getKey();
    if (key_pressed && isdigit(key_pressed)) {
      state = PASSWD_ENTERING;
    }
    if (scanCardID() == false) {      /*If no card, return*/
```

```
      return;
    }
    if (known_card_count > 0) {       /*If there even are ANY CARDS in the system*/
      int card_pos = isCardStored(act_card_id);
      if (card_pos == 0) {            /*Detected is MASTER card*/
        state = PROGRAMMING_CARD;
        break;
      } else if (card_pos > 0) {      /*Detected card is USER card*/
        state = UNLOCKED;
        break;
      } else {                        /*Detected card is NOT IN the system*/
        state = DENIED;
        break;
      }
    }
    break;
    /***************       UNLOCKED       **************/
  case UNLOCKED: /*System is unlocked*/
    if (lastState != UNLOCKED) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = UNLOCKED;
      UNLOCK();
      pprint("-----ACCESS----------GRANTED----");
      soundSuccess();
      resetFailedAttempts();
    }
    key_pressed = kp.getKey();
    if (key_pressed && (key_pressed == '#')) { /*Lock the system with keypad*/
      state = LOCKED;
      break;
    }
    /*Lock the system with any KNOWN card*/
    if ((scanCardID() == true) && (isCardStored(act_card_id) > -1)) {
      state = LOCKED;
      break;
    }
    break;
    /***************       PASSWD       **************/
  case PASSWD_ENTERING: /*System is locked, user has to authentify by password*/
    if (lastState != PASSWD_ENTERING) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = PASSWD_ENTERING;
      pprint("Enter Password");
      soundButton();
      lcdBottomPrint(passwd_tmp_digit_count, '*');
      password_tmp[passwd_tmp_digit_count] = key_pressed;
      passwd_tmp_digit_count++;
    }

    key_pressed = kp.getKey();
    if (key_pressed) {           /*Clasic password verifying*/
      soundButton();
      if (key_pressed == 'C') {
        if (passwd_tmp_digit_count > 0) {
          passwd_tmp_digit_count--;
          lcdBottomPrint(passwd_tmp_digit_count, ' ');
          password_tmp[passwd_tmp_digit_count] = '\0';
        } else {
          state = LOCKED;
        }
      } else if ((key_pressed == 'A') && (passwd_tmp_digit_count < PASSWD_MAX_LEN)) {
        if (isPasswordCorrect()) {
          state = UNLOCKED;
        } else {
          state = DENIED;
        }
      } else if (isdigit(key_pressed)) {
        lcdBottomPrint(passwd_tmp_digit_count, '*');
        password_tmp[passwd_tmp_digit_count] = key_pressed;
        passwd_tmp_digit_count++;
      }
    }
    break;
    /***************       PROGRAMMING_CARD       **************/
  case PROGRAMMING_CARD:  /*Adding/removing cards to/from system*/
    if (lastState != PROGRAMMING_CARD) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = PROGRAMMING_CARD;
      pprint("PROGRAM MODE    ADD/REMOVE CARDS");
    }
    while (true) {                          /*Waiting to detect some card*/
      programmingModeSigns(true);           /*Turns on and manages programming mode signals*/
      key_pressed = kp.getKey();
      if (scanCardID() == true) {           /*Some card detected*/
```

```
        card_pos = isCardStored(act_card_id); /*Is card in system? and what kind of card?*/
        if (card_pos > 0) {                   /*Card IS IN the system && it IS NOT a MASTER, REMOVE IT*/
          pprint("--CARD REMOVED--");
          failed_attempts = 0;
          soundButton();
          delay(1000);
          removeTag(card_pos);
          break;
        } else if (card_pos < 0) {            /*card IS NOT IN the system, ADD IT*/
          pprint("---CARD ADDED---");
          failed_attempts = 0;
          soundButton();
          delay(1000);
          known_cards[known_card_count] = strdup(act_card_id.c_str());
          known_card_count++;
          break;
        } else if (card_pos == 0) {           /*Card IS IN the system && it IS a MASTER, REMOVE IT*/
          pprint("MASTER CARD LOSTSCAN NEW MASTER!");
          soundFail();
          delay(1500);
          while (true) {        /*System cannot be wtihout MASTER, wait for user to SCAN A NEW ONE*/
            if (scanCardID() == true) {
              pprint("MASTER CARD SET");
              soundButton();
              delay(1000);
              card_pos = isCardStored(act_card_id);
              /*New MASTER card is already in the system as USER card, FIRST remove it as USER card*/
              if (card_pos > 0) {
                removeTag(card_pos);
              }
              free(known_cards[0]);             /*Removes current MASTER card from system*/
              known_cards[0] = strdup(act_card_id.c_str());
              break;
            }
          }
          break;
        }
      } else if (key_pressed == '#') {
        state = LOCKED;
        break;
      }
    }
    state = LOCKED;
    break;
    /***************       PROGRAMMING_PASSWD       ***************/
  case PROGRAMMING_PASSWD: /*Changging password stored in the system*/
    if (lastState != PROGRAMMING_PASSWD) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = PROGRAMMING_PASSWD;
      pprint("ENTER OLD PASSWD");
      failed_attempts = 0;
    }
    programmingModeSigns(false); /*Signalization of programming mode WITHOUT sound=false*/
    key_pressed = kp.getKey();
    if (key_pressed) {
      if (key_pressed == 'C') {
        if (passwd_tmp_digit_count > 0) { /*Clear one '*' from display and passwd after pressing 'C'*/
          soundButton();
          passwd_tmp_digit_count--;
          lcdBottomPrint(passwd_tmp_digit_count, ' ');
          password_tmp[passwd_tmp_digit_count] = '\0';
        }
      } else if (key_pressed == 'A') {    /*Verify the passwd */
        if (passwd_change_flag) {         /*If old/stored passwd is verified this flag is true*/
          soundButton();
          delay(300);
          soundSuccess();
          memset(password_stored, 0, strlen(password_stored) * sizeof(char));
          strcpy(password_stored, password_tmp);
          pprint("NEW PASSWD IS:  " + (String) password_stored);
          delay(2000);
          passwd_change_flag = !passwd_change_flag;
          state = UNLOCKED;
          break;
        } else if (isPasswordCorrect()) {    /*Old/Stored passwd is successfuly verified, set the flag*/
          pprint("ENTER NEW PASS  AND PRESS 'A' ");
          soundButton();
          delay(300);
          soundSuccess();
          delay(2500);
          pprint("NEW PASSWORD:");
          passwd_change_flag = !passwd_change_flag;
          break;
        } else {
```

```
          pprint("WRONG OLD PASSWD   TRY AGAIN!");
          soundFail();
          delay(2000);
          failed_attempts++;
          if (failed_attempts > MAX_ATTEMPTS) {
            state = ALARM;
            break;
          }
          pprint("ENTER OLD PASSWD");
          lcdBottomPrint(0, '#');            /*clears the bottom part of display*/
        }
      } else if (isdigit(key_pressed)) {    /*Some digit of the passwd is given, acknowledge it*/
        lcdBottomPrint(passwd_tmp_digit_count, '*');
        soundButton();
        password_tmp[passwd_tmp_digit_count] = key_pressed;
        passwd_tmp_digit_count++;;
      } else if (key_pressed == '*') {
        state = UNLOCKED;
        break;
      }
    }
    break;
    /***************       DENIED       ***************/
  case DENIED: /*Wrong password or card was presented*/
    if (lastState != DENIED) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = DENIED;
      pprint("-----ACCESS----------DENIED-----");
      soundFail();
      delay(1500);
    }
    failed_attempts++;
    if (failed_attempts > MAX_ATTEMPTS) {
      state = ALARM;
      break;
    }
    state = LOCKED;
    break;
    /***************       ALARM       ***************/
  case ALARM:
    if (lastState != ALARM) {
      lastState = ALARM;
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      pprint("-TOO MANY TRIES-ALERTING POLICE!!");
      soundAlarm(4000);
    }
    LOCK();
    resetFailedAttempts();
    state = LOCKED;
    break;
    /***************       SYSTEM_RESETT       ***************/
  case SYSTEM_RESETT: /*RESET or RESTART system state*/
    if (lastState != SYSTEM_RESETT) {
      // Serial.println("ENTERING STATE: " + String(state_name[state]));
      lastState = SYSTEM_RESETT;
      if (arduino_resart_flag == true) {
        pprint(" ENTER SPECIAL  RESTART PASSWD");
      } else {
        pprint("  ENTER FACTORY   RESET PASSWD");
      }
      delay(3000);
      pprint("EMERGENCY PASSWD");
      lcdBottomPrint(0, '#');
    }
    programmingModeSigns(false);
    key_pressed = kp.getKey();
    if (key_pressed) {
      if ((key_pressed == 'C') && (passwd_tmp_digit_count > 0)) {
        soundButton();
        passwd_tmp_digit_count--;
        lcdBottomPrint(passwd_tmp_digit_count, ' ');
        password_tmp[passwd_tmp_digit_count] = '\0';
      } else if (key_pressed == 'A') {          /*Verify emergency passwd before restart/reset*/
        int result = strncmp(password_tmp, password_emergency, strlen(password_emergency));
        if (result == 0) {
          pprint("    PASSWORD        ACCEPTED");
          delay(2000);
          resetFailedAttempts();
          resetPasswordTmpCounter();
          if (arduino_resart_flag == true) {
            pprint("   SYSTEM IS      RESTARTING!!!");
            delay(2000);
            arduino_resart_flag = false;
            resetFunc();
```

```
            return;
          }
          pprint(" RESETTING TO   FACTORY SETTINGS");
          soundAlarm(4000);
          pprint("SCAN NEW MASTER NEW PASSWD: 1234");
          removeAllCards();
          memset(password_stored, 0, strlen(password_stored) * sizeof(char));
          char tmpPass[] = {'1','2','3','4','\0'};
          memcpy(password_stored, tmpPass, sizeof(tmpPass));
          /*System was reset to factory settings and user HAS TO scan a new master card! */
          while (true) {
            if (scanCardID() == true) {a
              known_cards[0] = strdup(act_card_id.c_str());
              known_card_count++;
              pprint("MASTER CARD SET");
              soundButton();
              delay(1000);
              state = LOCKED;
              break;
            }
          }
          break;
        } else {
          pprint("WRONG EMERGENCY PASSWD TRY AGAIN");
          soundFail();
          delay(2000);
          resetPasswordTmpCounter();
          failed_attempts++;
          if (failed_attempts > MAX_ATTEMPTS) {
            state = ALARM;
            break;
          }
          pprint("EMEGENCY PASSWD:");
          lcdBottomPrint(0, '#');
        }
      } else if((key_pressed == '#')||(key_pressed == 'D')) {
        state = LOCKED;
        break;
      } else if (isdigit(key_pressed)) {
        lcdBottomPrint(passwd_tmp_digit_count, '*');
        soundButton();
        password_tmp[passwd_tmp_digit_count] = key_pressed;
        passwd_tmp_digit_count++;
      }
    }
    break;
    /***************       OTHER       **************/
  case OTHER:
    break;
  default:
    break;
  }
}
```