UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Smart key locker

propusă de

Andreea-Larisa Avădănei(Sfîrnaciuc)

Sesiunea: iulie, 2019

Coordonator științific

Lect. Dr. Anca Ignat

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Smart key locker

Andreea-Larisa Avădănei(Sfîrnaciuc)

Sesiunea: iulie, 2019

Coordonator științific

Lect. Dr. Anca Ignat

Avizat,

Îndrumător lucrare de licență,

Lect. Dr. Anca Ignat.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Avădănei(Sfîrnaciuc) Andreea-Larisa** domiciliat în **România, jud. Iași, mun. Iași, str. Ursulea, nr. 8**, născut la data de **28 martie 1994**, identificat prin CNP **2940328226741**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2019, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Smart key locker** elaborată sub îndrumarea domnului **Lect. Dr. Anca Ignat**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Smart key locker**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea ”Alexandru-Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Andreea-Larisa Avădănei(Sfîrnaciuc)**

Data: ..... Semnătura: .....

Contents

Motivation

A few years ago I had this English class were our teacher had this lesson about Smart houses. I was really fascinated about how you actually can make your house alive. In other words my curiosity about Artificial Intelligence started since then and that's why I have started to pay more attention to all courses which had something in common with this subject. During the college I didn't had the right opportunity to put everything I have learned together and somehow my dream to build something similar was put on hold. With some luck, last year an old dream of mine got fulfilled and I moved into a house with small garden. In other words now I had my house, but not a smart one so, I thought if a really old dream came true why shouldn't I continue with the other dream? Because I am not really close to the city, I thought that one of the best first project which I should start should be a Smart Lock Key. This idea would be great first of all for protection and also a great start to make my house alive.

Introduction

The theme of license is developing a smart key locker with two features of verification.

One is the classic way and most common method, open the gate using a pin. Seconds method I have inspired from the google with the two step verification: the gate is opened by using a pin and then wait for a second pin which is sent via personal email address set.

This type of locker developed can not be found in the market the way I have made it. In order to build it I have developed it from the stage where i needed to combine the hardware components which I bought: moteino, WiFi receiver, numeric pad and display.

In first chapter " Presentation of software technologies" in section 1.1 I will detail the principle of running the Arduino IDE application, how to communicate with an Arduino device, and the benefits of using it. In section 1.2. I will briefly present the Serial Monitor, and its need in developing a project. In sectionr 1.3. I will analyze the libraries used, and explain for what are their usage.

In the second chapter "Presenting the Hardware Technologies Used", there are 4 sections describing how to use and operate the components that make up the circuit of my lock key.

In the 3rd chapter, "Application Software Architecture" describes how the user communicates with the lock via the LCD and the numeric pad.

In the 4th chapter, "Application Hardware Architecture" describes the steps taken in building the lock, starting from the purchase of the required components, and their assembling.

In the 5th chapter I described some problems encountered, I presented the cause of the problems and how I solved them.

Short presentation of project idea

In order to offer a short overview of my project I will explain from two point of view the project idea.

1. A black box overview. What user is seeing and what operation should make with the locker.

2. A white box overview. This is regarding what is inside the code, what each class stands for.

Black box overview

When the user will operate with the locker first time will be asked to introduce a pin. The pins he can introduce is the ”Gate Pin” which will open the gate and the other one is” Master Pin” which will open a menu for the setting of the locker.

The opening of the gate can be made through two ways: - one is opening the gate by introducing only one pin which is memorised in persistent memory. - the second one requires 2 pins. One is from the persistent memory and the second is the one sent by the locker via email.

The user can choose what kind of the method wants for its own locker by using master pin and entering in the settings. In the settings menu he can change the gate pin or master pin, enable the email in order to have a two step verification or disable it and use only one pin.

White box overview

The locker is build up from two main components which makes possible to have all functions available from the black box resume.

The main node is Moteino Mega. In Moteino mega I have the following classes:

uart inc - stands for the communication between ESP8266 WiFi and Moteino; Storage - level access, write to flash memory; Security - write everything we need, pin,

email; LogTrace - prints all traces through the serial; keymapping - code for numeric pad; GUI - shows everything what is seen on the display;

The other node is ESP8266 WiFi. In WiFi i have the following classes: WiFi Conn - the all connection to the WiFi MailSender- to send the emails LogTrace - prints all traces through the serial; uart inc - scommunication between ESP8266 Wifi and Moteino;

Chapter 1

Presentation of software technologies

1.1 Arduino Software IDE

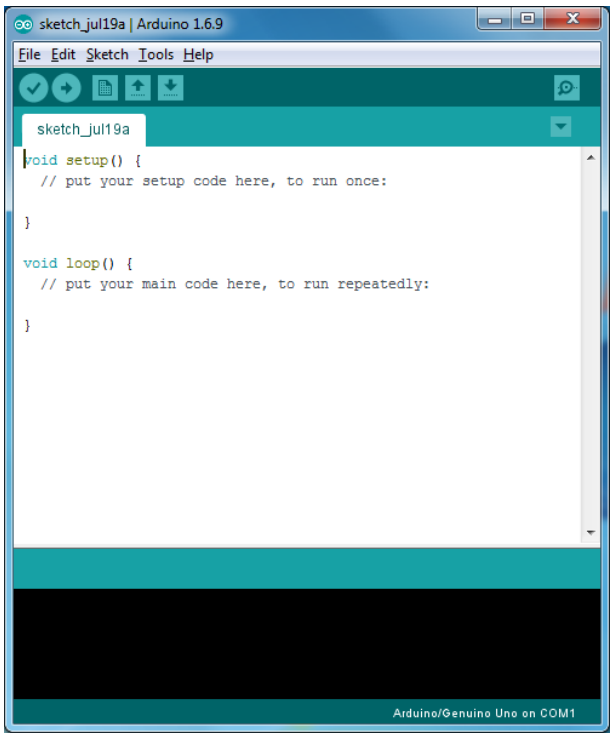
Arduino programs can be written in any programming language while using a compiler that produces a machine code binary.

Arduino IDE (Integrated Development Environment) [1] is an open-source cross-platform (multiplatform) application written in the Java programming language. Being multiplatform, it means it is produced in several variants and can be used on multiple hardware platforms or operating systems.

The program includes a code editor with predefined functions for easier use (e.g., Auto Spacing, matching matches, highlighting keywords / keywords, etc.). A program written in the Arduino IDE is called sketch and is the extension.

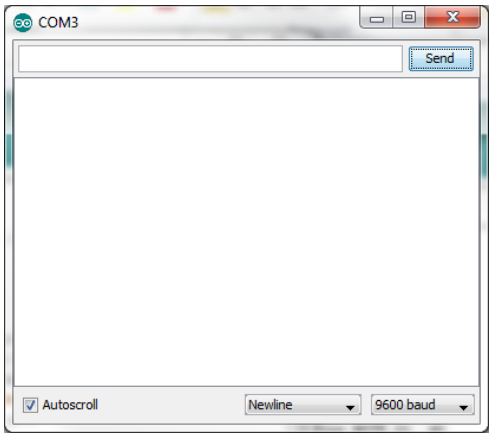
The supported programming languages are C and C ++ and use the special rules for organizing the code. An Arduino sketch is based on 2 features: • setting (): installs only once at the beginning of the program to initiate the settings • Loop (): Run until the Arduino board is powered off

Following the compilation of the source code and the use of the programming tools included in the GNU tool-chain [4], Arduino sends the AVRDUDE [5] command to convert the executable code into a hexadecimal encoded text file, then be uploaded to the Moteino device a load schedule. Source code accuracy can be verified before uploading it using the Compile button, or verified and uploaded using the Upload button.



1.2 Serial Monitor

Another feature of this IDE is the serial monitor. When used, a new window opens which communicates by sending / receiving serial data. To be used, we need to have a connected Moteino device and generally serves as a debugger in the created sketches.

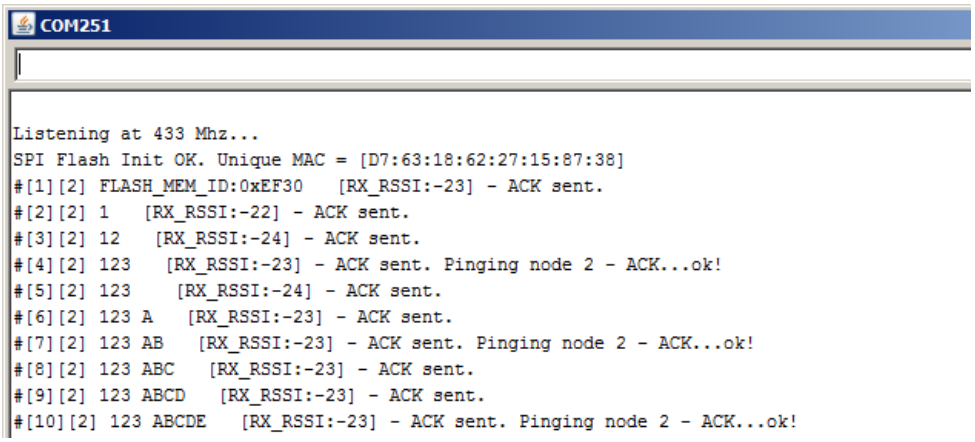


1.3 Libraries used

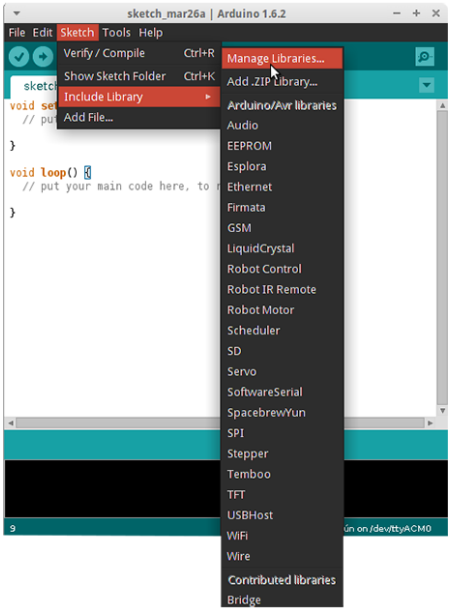
Arduino IDE has a number of predefined bookstores that make it easier to control the hardware of the module, with mostly deployed methods for what is retrieving

or playing such a module. In order to use MoteinoMEGA-USB borad I will use the following 2 libraries: RFM69, SPIFlash.

Once the libraries and the core are installed the baud choose should be 115200 and No Line Ending settings at the bottom. For the gateway this is what should be seen:



In order to accomplish the project, it was necessary to download and import bookshops other than those already existing:



1.3.1 RFM69

The easiest way to get started is with the well documented and supported Moteino microcontroller platform which is easily programmable from the Arduino IDE. This includes the Moteino, MoteinoUSB and MoteinoMEGA. RFM69 transceivers ware exten-

sively tested on Moteinos for the purpose of building internet of things (IoT) devices that can be controlled wirelessly.

Features

- easy to use API with a few simple functions for basic usage
- 61 bytes max message length
- customizable transmit power (32 levels) for low-power transmission control
- sleep function for power saving
- automatic ACKs with the sendWithRetry() function
- hardware 128bit AES encryption
- hardware preamble, synch recognition and CRC check
- digital RSSI can be read at any time with readRSSI()

1.3.2 SPIFlash

Arduino/Moteino library for read/write access to SPI flash memory chips. This works with 256byte/page SPI flash memory such as the 4MBIT W25X40CLSNIG used on Moteino for data storage and wireless programming. Minimal modifications should allow chips that have different page size to work. This library was primarily developed to enable safe wireless programming on Moteino nodes and Moteino based applications such as the SwitchMote. Dualoptiboot and RFM69-OTA Wireless-Programming library are required to be able to wirelessly re-flash a remote Moteino.

1.3.3 Adafruit-GFX library

This is the core graphics library for all our displays, providing a common set of graphics primitives (points, lines, circles, etc.).

1.3.4 ESP8266 WiFi

ESP8266 Arduino core comes with libraries to communicate over WiFi using TCP and UDP, set up HTTP, mDNS, SSDP, and DNS servers, do OTA updates, use a file system in flash memory,

1.3.5 WiFiClientSecure

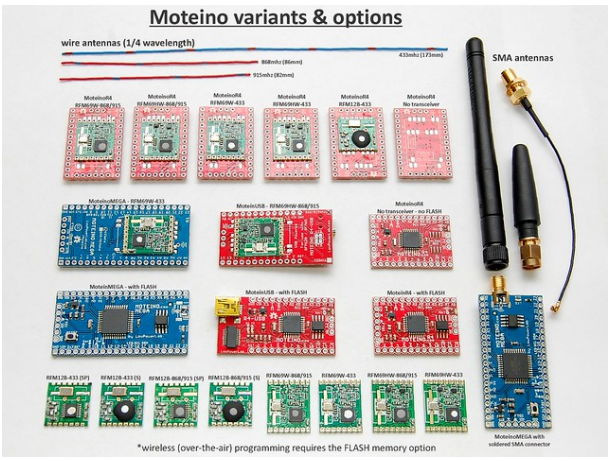
The WiFiClientSecure class implements support for secure connections using TLS (SSL). It inherits from WiFiClient and thus implements a superset of that class' interface.

Chapter 2

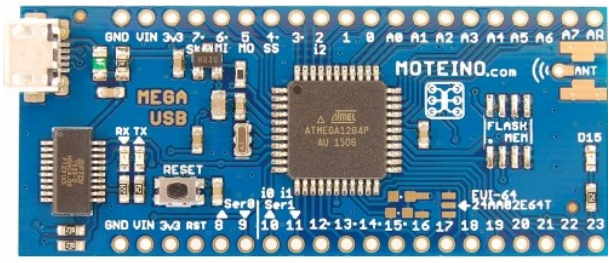
Presentation of hardware technologies

2.1 MoteinoMEGA-USB

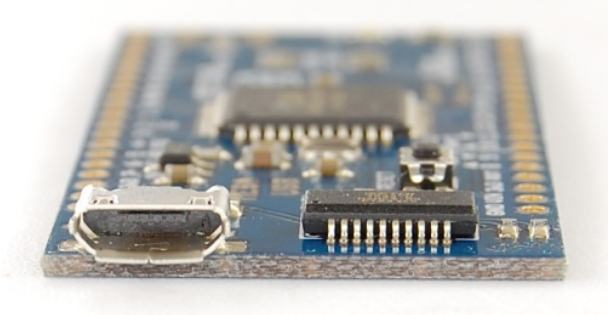
There is a wide range of Moteino devices such as: MoteinoR4, MoteinoUSB, MoteinoR4 with Flash, MoteinoMega-with Flash ect., but I have opted for MoteinoMega-USB



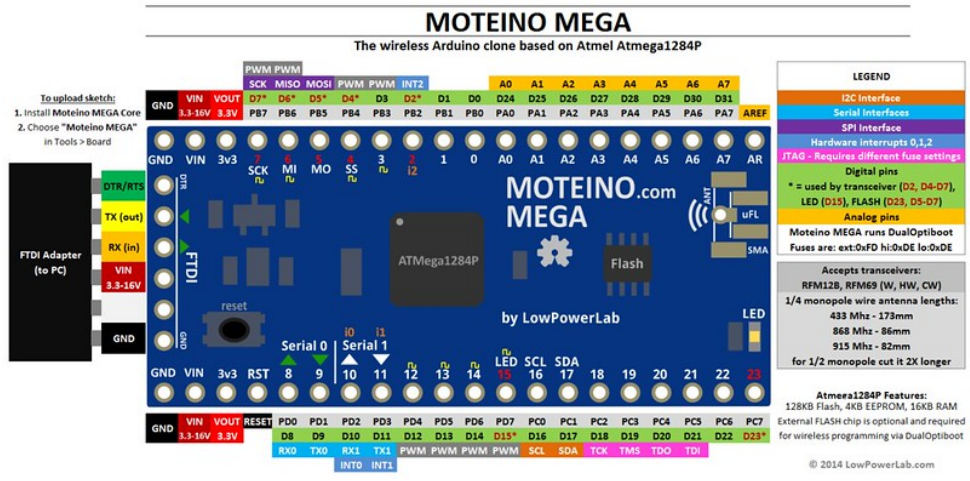
The MoteinoMEGA uses the Atmega1284p microcontroller. It still runs at 3.3V/16Mhz but has 128KB of internal flash, 16KB of RAM, 4KB EEPROM, 2x hardware serial ports, 8x PWM pins and bunch more GPIO pins. Still a small board at just 1.0x2.0 and bread-board friendly. Comes with the same DualOptiboot bootloader making it wireless pro-gramming capable, and can take up to 16V of input on the VIN pins.



MoteinoMEGA-USB is designed to be used as a main gateway for a network of Moteino nodes, without the need of an additional FTDI adapter.



Pinout



2.1.1 Why did I choose Moteino?

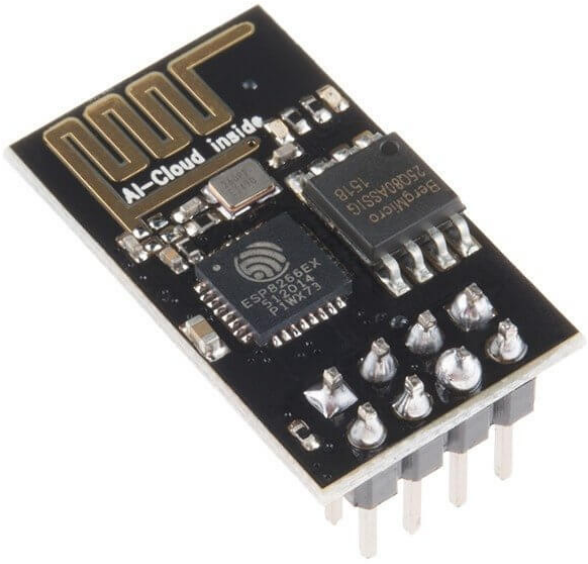
Moteino is designed as a compact, highly customizable and affordable development platform, suitable for IoT, home automation and long range wireless projects.

Features that set Moteino apart:

- small and light modular design fits in tiny enclosures
- flexible configurations allow several wireless transceivers to be used
- true ultra low power: with just 2uA achievable in deep sleep mode
- Watchdog sleep mode is at 6uA (periodic wake).
- wirelessly (aka OTA) programmable: be able to re-flash it without wires
- easy to use from the familiar Arduino IDE.

2.2 Serial WIFI Wireless ESP8266

In order to be able to develop 2 step door unlock I have used Serial WIFI Wireless ESP8266 Transceiver. It is a complete and autonomous Wi-Fi networking solution that can transport software applications. Has an embedded cache memory which improves system performance and reduce memory requirements.



2.3 OLED Display 0.91

In order to make the user interface more friendly and interactive I have decided to use a small size display which comes with the SSD1306 chipset, and has a 128x32 resolution.The standard of the IIC serial interface allows this screen to be connected to MCU development boards such as Arduino, Raspberry Pi, C51 series, etc.

In special I have chosen this OLED because it has a very wide viewing angle of 160 degrees. Also, the OLED technology on which this display is based will provide a

high contrast and at the same time the power consumption will be very low.

When the application is powered up on the Display will appear the text ”Please enter Pin:” Where we can introduce to types of pin: one if the ”Gate pin” which is going to open up the Gate and the other one is the ”Master Pin” where modification can be made.

2.3.1 Introducing Gate Pin

One Step verification

When one step verification pin setup is chosen the Gate pin memorised in setup needs to be introduced.

Two Step verification

When the two step verification setup is chosen the same Gate Pin memorised from One step verification need to be introduced. After this we need to verify the our personal email for the second pin generated by the Lock door key in order to open the gate

```
void gUI::DrawMainMenu(){
  logtrace_print(T_ALL,__func__);
  myDisplay->clearDisplay();
  myDisplay->setCursor(0, 0);     // Start at top-left corner
  myDisplay->println(F("Please enter Pin:"));
  myDisplay->display();

  char key = '0';
  uint8_t index = 0;
  char pin[MAX_PIN];
  memset(pin,0,MAX_PIN);

  while(key!='#'){
    key = myKeypad->getKey();
    if(key){
      if(index >= 9){
        pin[index]='#';
        break;
      }
      else{
        pin[index] = key;
        index++;
        myDisplay->print(key);
        myDisplay->display();
      }
    }
  }
   if( mySecurity.ValidateMasterPin(pin,index) )
     DrawSetupMainMenu();
   else{
      if( mySecurity.ValidateGatePin(pin,index) ){
       DrawGateOpen();
       DrawMainMenu();
      }
      else{
       DrawMainMenu();
      }
  }
}
```

2.3.2 Introducing Master Pin

When the Master Pin is introduced the following Menu is going to be displayed:

1. Master Pin Setup
2. Gate Pin Setup
3. Email Setup
4. Print Current Config

```
void gUI::DrawSetupMainMenu(){
  logtrace_print(T_ALL,__func__);
  myDisplay->clearDisplay();
  myDisplay->setCursor(0, 0);     // Start at top-left corner
  myDisplay->println(F("1. Master Pin Setup"));
  myDisplay->println(F("2. Gate Pin Setup"));
  myDisplay->println(F("3. Email Setup"));
  myDisplay->println(F("4. Print Current Config"));
  myDisplay->display();

  char key = '0';
  while(1){
    while( key == '0' ){
      key = myKeypad->getKey();
      if( key == '1' )
        DrawPinMainMenu(PIN_TYPE_MASTER);
      if( key == '2' )
        DrawPinMainMenu(PIN_TYPE_GATE);
      if( key == '3' )
        DrawEmailMainMenu();
      //if( key == '4' )
        //DrawCurrentConfig()
      key = '0';
    }
  }

}
```

Master Pin Setup and Gate Pin Setup

When key 1 is pressed the application enters in Master Pin Setup we can do the following operations:

1. Setup a new Pin
2. Print current Pin
3. Remove current Pin
9. Main menu

The same Menu will appear when key 2 is pressed

```
// draw SetupMainMenu
void gUI::DrawPinMainMenu(uint8_t pinType ){
  logtrace_print(T_ALL,__func__);
  myDisplay->clearDisplay();
  myDisplay->setCursor(0, 0);     // Start at top-left corner
  myDisplay->println(F("1. Setup a new Pin"));
  myDisplay->println(F("2. Print current Pin"));
  myDisplay->println(F("3. Remove current Pin"));
  myDisplay->println(F("9. Main menu"));
  myDisplay->display();

  char key = '0';
  while(1){
    while( key == '0' ){
      key = myKeypad->getKey();
      if( key == '1' )
        DrawSetPinMenu(pinType);
      if( key == '2' )
        DrawDisplayCurrentPinMenu(pinType);
      if( key == '9' )
        DrawMainMenu();
      key = '0';
    }
  }
}
```

Email Setup

When key 3 is pressed, the application enters in the email setup and the bellow operations can be done.

1. Setup Email
2. Print Email
3. Disable Email Authentication
9. Main menu

In this section we can setup at which email address we would like to send the second step verification, print out what is the current email address and password or disable this 2 step verification via email.

Print Current Config

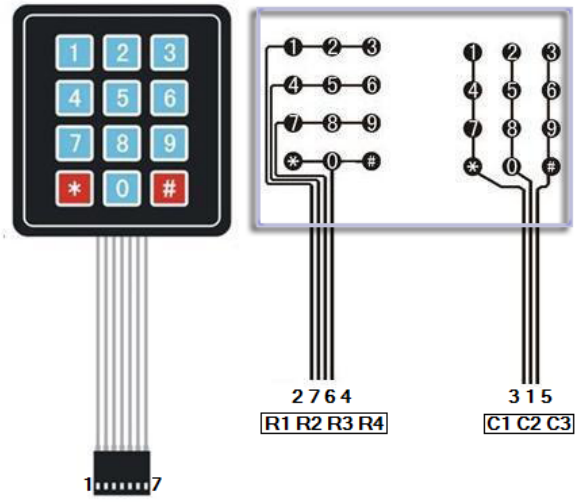
When key 4 is pressed via display we can verify what is the current configuration for the smart key locker.(one step verification or two step verification)

2.4 Numeric Pad

In order to interact with the HW in a easy way I have decided to use a 3x4 KEYPAD.

2.4.1 How keypad works

The buttons on a keypad are arranged in rows and columns. A 3X4 keypad has 4 rows and 3 columns. Beneath each key is a membrane switch. Each switch in a row is connected to the other switches in the row by a conductive trace underneath the pad. Each switch in a column is connected the same way – one side of the switch is connected to all of the other switches in that column by a conductive trace. Each row and column is brought out to a single pin, for a total of 7 pins on a 3X4 keypad:



The button presses can be translated into electrical data for use with a microcontroller. The wiring up the keypad to the Moteino in the following manner:

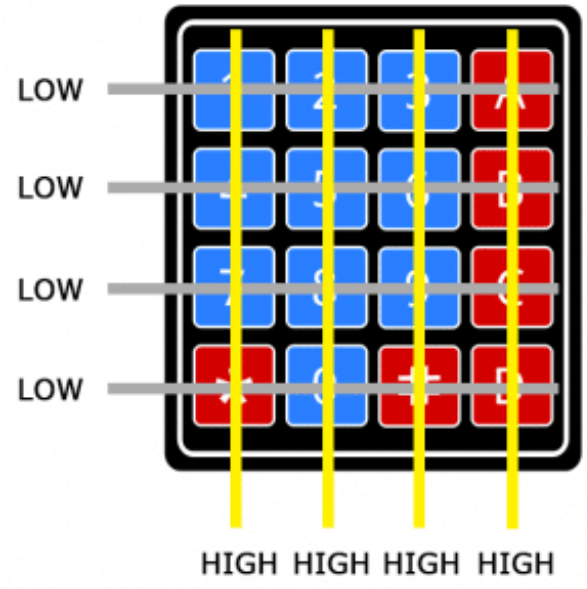
Keypad pin 1 to Moteino digital 3 //C2
Keypad pin 2 to Moteino digital 5 //R1
Keypad pin 3 to Moteino digital 2 //C1
Keypad pin 4 to Moteino digital 8 //R4
Keypad pin 5 to Moteino digital 4 //C3
Keypad pin 6 to Moteino digital 7 //R3
Keypad pin 7 to Moteino digital 6 //R2

```
#include <Keypad.h>

static const byte ROWS = 4; //four rows
static const byte COLS = 3; //three columns
static char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
static byte rowPins[ROWS] = {21, 12, 13, 19};
static byte colPins[COLS] = {20, 22, 18};

static Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

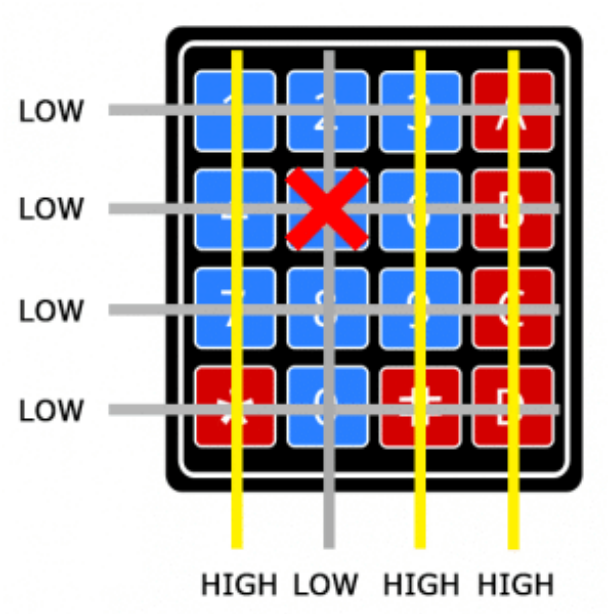
The Moteino detects which button is pressed by detecting the row and column pin that's connected to the button.

2.4.2 Logic steps

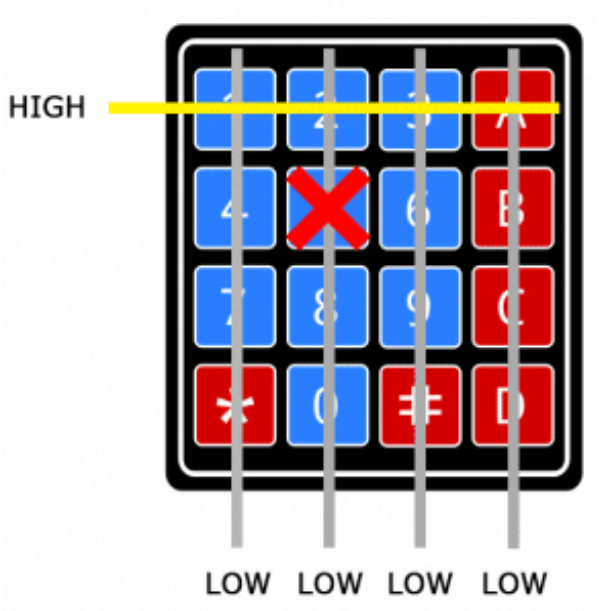
1. When no buttons are pressed, all of the column pins are held HIGH, and all of the row pins are held LOW



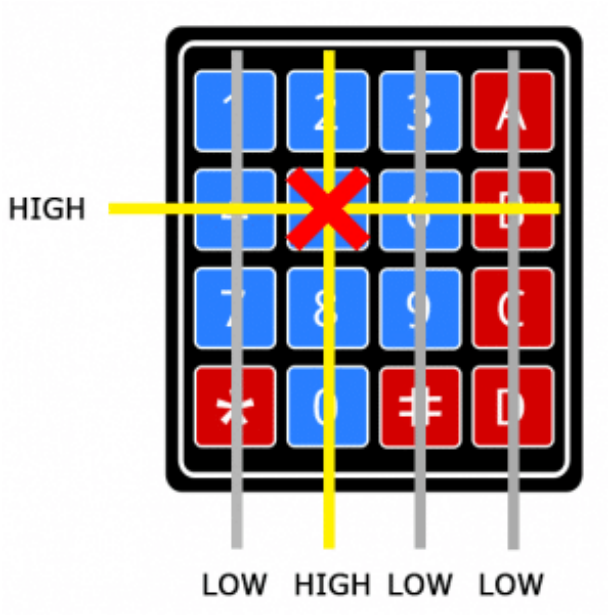
2. When a button is pressed, the column pin is pulled LOW since the current from the HIGH column flows to the LOW row pin



3. The Arduino now knows which column the button is in, so now it just needs to find the row the button is in. It does this by switching each one of the row pins HIGH, and at the same time reading all of the column pins to detect which column pin returns to HIGH:

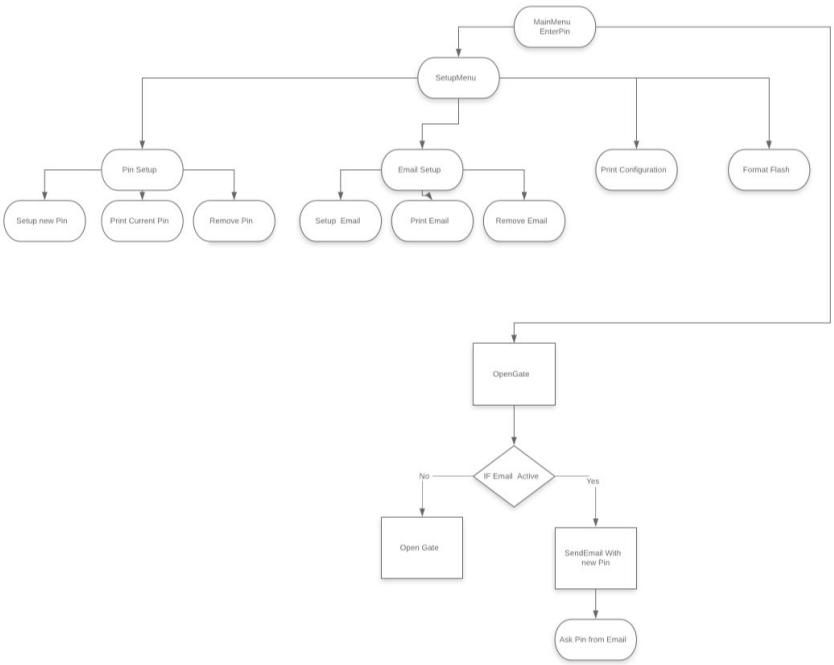


4. When the column pin goes HIGH again, the Arduino has found the row pin that is connected to the button:


HIGH
LOW
HIGH
LOW
LOW

Chapter 3

Application software architecture



The use of the technological modules for the desired purposes would not be possible without programming. The locking key control is done via the numeric pad and the LCD.

Through the numeric pad the data introduced is processed and transposes into the functionality of the lock.

Main Menu- when the locker is on via display the text ”Enter a pin” will be displayed. Depending on which pin we enter, master pin or gate pin, will access two different functions.

SetupMenu is going to be accessed by the ”Master Pin”. In this menu depending which number we send via numeric pad we will access one setting. - Key 1 and 2

stands for the Pin Setup (Master, Gate) where we can do the operation of setting a new pin, print the current pin or remove the current pin. - Key 3 stands for the Email Setup where we can do the operation of setting a new Email, print the email where we send the pin, remove the email. - Key 4 will print the current configuration.

OpenGate is going to be accessed by the ”Gate pin”. This stage of opening the gate depends if the email is active or not. In case is the email is disabled, the gate will be opened. In the other scenario if the email is active, the locker will be sending a new pin generated to the email address memorised. With the new pin obtained from the email the gate will be opened.

Chapter 4

The hardware architecture of the application

To make this locker to open automatically the gate, I had to carefully choose each piece and each component required to put it into operation.

Following the browse of several websites and also what I would like to create in the final results of the locker, I made a decision and made a list with the necessary components.

Each component has a well-established role, with less or greater utility compared to the other modules in the circuit. Some components are entirely dependent on the presence of others (as is the case with WiFi ESP8266, which receives the input data through serial from the MoteinoMega).

I used the MoteinoMega because it has enough pins to connect all the necessary components.

The usefulness of the components would not exist without wiring them accordingly to the hardware instructions. This is done using 10 cm wires and a usb cable connected to the usb laptop port.

Problems encountered

Being the first time working on a project of this kind, the problems started to show up when it was less expected. The first big issue which I had it was regarding WiFi. The WiFi receiver needs 3.3V, but¿ 300 amps and my moteino gives around 250 amps. In the end I connected the WiFi to Moteino and made it as a master and through an USB cable I connected it at the laptop.

Another issue which I had it was regarding how I can send an email and how can I connect to the WiFi. These two issues I solved them by using the libraries ”WiFi-ClientSecure.h” and ”ESP8266WiFi”.

Email sending

```
bool MailSender::GetSMTPAnswer(WiFiClientSecure &SMTP_Client, const String &answer, uint16_t timeO
  logtrace_print(T_ALL,__func__);
  uint32_t ts = millis();
  while (!SMTP_Client.available())
  {
    if(millis() > (ts + timeOut)) {
      last_error = "SMTP Response TIMEOUT!";
      return false;
    }
  }
  serverAnswer = SMTP_Client.readStringUntil('\n');
  logtrace_print(T_DEBUG,serverAnswer.c_str());
  if (answer && serverAnswer.indexOf(answer) == -1)
    return false;
  return true;
}

void MailSender::Init(){
  logtrace_print(T_ALL,__func__);
  bInitDone = true;
}

char* MailSender::error_translator( const char* answer_expected ){
  if( "220" == answer_expected )
    return "Connection error";
  if( "250" == answer_expected )
    return "Identification error - Sending message error";
  if( "235" == answer_expected )
    return "SMTP AUTH error";
  if( "354" == answer_expected )
    return "SMTP DATA error";
  if( "221" == answer_expected )
    return "SMTP QUIT error";
}

bool MailSender::SendToSMTP(WiFiClientSecure &SMTP_Client,  const String message ){
  logtrace_print(T_ALL,__func__);
  logtrace_print(T_DEBUG,"sending: %s",message.c_str());
  SMTP_Client.println(message);
}


bool MailSender::SendToSMTP_W(WiFiClientSecure &SMTP_Client,  const String message, const String &answer_expected, uint16_t timeOut ){
  logtrace_print(T_ALL,__func__);

  logtrace_print(T_DEBUG,"sending: %s",message.c_str());
  SMTP_Client.println(message);

  logtrace_print(T_DEBUG,"expecting: %s",answer_expected.c_str());
  if(!GetSMTPAnswer(SMTP_Client, answer_expected)){
    last_error = error_translator(answer_expected.c_str());
    logtrace_print(T_DEBUG,last_error );
  }
```

```
  logtrace_print(T_DEBUG,"expecting: %s",answer_expected.c_str());
  if(!GetSMTPAnswer(SMTP_Client, answer_expected)){
    last_error = error_translator(answer_expected.c_str());
    logtrace_print(T_DEBUG,last_error );
  }
}

bool MailSender::Send( const String &message ){
  logtrace_print(T_ALL,__func__);
  return Send(EMAIL_DEST,message);
}


bool MailSender::Send( const String &dest, const String &message ){
  logtrace_print(T_ALL,__func__);
```

WiFi Connection

```
void WiFiConn::Init(const char* ID, const char* pass, uint16_t timeout){
  logtrace_print(T_ALL,__func__);
  if( NULL == ID || NULL == pass )
    return;

  ssID = new char[strlen(ID)];
  memset(ssID,0,strlen(ID));
  logtrace_print(T_ALL,"%s, %d",ID,strlen(ID));
  //strcpy(ssID, ID);
  memcpy(ssID,ID,strlen(ID)-1);

  password = new char[strlen(pass)];
  memset(password,0,strlen(pass));
  logtrace_print(T_ALL,"%s, %d",pass,strlen(pass));
  //strcpy(password,pass);
  memcpy(password,pass,strlen(pass)-1);

  retry_time = timeout;
  bInitDone = true;
}
bool WiFiConn::Connect(){
  logtrace_print(T_ALL,__func__);
  uint8_t counter = 0;
  WiFi.begin(ssID,password);
  logtrace_print(T_ALL,"ssID: %s\nPass: %s",ssID,password );

  logtrace_print(T_DEBUG,"Trying to connect to: %s",ssID );
  while(WiFi.status() != WL_CONNECTED && counter++ < WiFi_MAX_WAITING_COUNTS ){
   delay(WiFi_CONN_WAIT_DELAY);
   logtrace_print(T_DEBUG," * ");
  }

  if( counter > WiFi_MAX_WAITING_COUNTS ){
   logtrace_print(T_INFO,"Unable to connect to %s\n",ssID);
   conn_status = CONN_TIMEOOUT;
   return false;
  }
  logtrace_print(T_INFO,"Connected to %s\n",ssID);
  conn_status = CONN_ON;

  IPAddress ip = WiFi.localIP();
  logtrace_print(T_ALL,ip.toString().c_str());

  return true;
}
```

```
void WiFiConn::Disconnect(){
  logtrace_print(T_ALL,__func__);
  WiFi.disconnect();
  conn_status = CONN_OFF;
}

uint8_t WiFiConn::GetStatus(){
  logtrace_print(T_ALL,__func__);
  return conn_status;
}

const char* WiFiConn::GetIP(){
  logtrace_print(T_ALL,__func__);
  IPAddress ip = WiFi.localIP();
  return ip.toString().c_str();
}

bool  WiFiConn::IsInit(){
  return bInitDone;
}
```

The third issue was how can I make the communication between the moteino and wifi nodes. This I got solved through making the Moteino wifi send and wait responses from each other.

```
#include <stdio.h>
#include <arduino.h>

#define   MAX_MSG_SIZE  255

#define   M_ACK          "[254]"
#define   M_TIMEOUT      "[253]"
#define   M_UNEXPECTED   "[252]"
#define   M_NOK          "[251]"

#define   M_PIN   "[001]"

#define   M_SMTP_INIT     "[100]"
#define   M_USERNAME      "[101]"
#define   M_PASSWORD      "[102]"
#define   M_SMTP_SERVER   "[103]"
#define   M_SMTP_PORT     "[104]"
#define   M_MAIL_SENDER   "[105]"
#define   M_MAIL_DEST     "[106]"
#define   M_SEND_MAIL     "[199]"

#define   M_WIFI_INIT       "[200]"
#define   M_WIFI_SID        "[201]"
#define   M_WIFI_PASS       "[202]"
#define   M_WIFI_CONNECT    "[299]"
#define   M_WIFI_DCONNECT   "[298]"

void get_message( String* message );
void send_ack( char* ack_type );
bool wait_for_next_message( char* answer, uint32_t timeout, char *message_content );
void pars_message( String message, char* message_header, char* message_content );

#endif
```

Conclusion and directions for future development

While working on this project I got to discover how fun and hard at the same time is it to start a project from the beginning without having loads of knowledge in the beginning. Through this project I have educated my style of working and also I have improved my abilities to solve my own blocking points. As I said in my motivation, I will not stop here with this project because I would like to use it on my own house.

What I would like to improve:

1. The password for Pin to be Unique. At the moment we can use the same pin for master and gate. The solving is very simple, when a new pin is added, master or gate, I will made an additional validation where I will check if the current pin introduced is not already set.

2. Another improvement which i find it absolutely necessary is the security of the password. I will add a cyclic redundancy check function.

3. In situations when the wrong pin is introduced an email with an alert should be sent.

4. In situations if the WiFi module has an error due to any reason, the email option should be sent disabled by default.

What I would like to add:

1. To add manually at which address I would like to send the PIN for the 2 step verification. At the moment the email address and password are hard-coded. This would be useful when is it needed to be used this smart key lock by the multiple family members.

2. To be able to visualise what is the current email address set also what is the current configuration. This will help to know what are the

3. To add a reset function to the WiFi module.

Bibliography

- https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
- https://lowpowerlab.com/guide/moteino/
- https://www.base64encode.org/
- https://github.com/adafruit/Adafruit-GFX-Library
- https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiF
  src
- http://www.mcl.hu/sites/default/files/IoT%20guide_0.pdf
- https://www.amazon.co.uk/Preptec-switch-Membrane-Arduino-keyboard/
  dp/B01MECWAW5
- https://www.electroschematics.com/12446/arduino-with-keypad/
- https://www.allaboutcircuits.com/projects/introduction-to-the-rfm69
- https://github.com/LowPowerLab/SPIFlash