Spring 2021

Electronic Locking Mechanism

Nicholas Tamburrino
njt26@uakron.edu

Electronic Locking Mechanism

Senior Design Project Final Report

Nicholas Tamburrino

Electrical and Electronic Engineering Technology

April 12, 2021

Advisor: Dr. Andrew Milks

Table of Contents

Abstract

This report contains information regarding the development and design of an electronic locking mechanism to be used for personnel access control. The lock provides a medium to low level of security and can be opened by authorized users via a radio-frequency identification fob or use of a numeric passcode. Many modern door locks similar to the one described above use solenoids and/or electromagnetic relays to control the lock. However, using these components can create vulnerabilities because they can be activated by unauthorized users using a sufficiently strong magnet. The purpose of the lock designed for this project was to eliminate the magnetic vulnerability by using an Arduino microcontroller to control a stepper motor to unlock the door. Using a microcontroller and a motor as the main control circuitry for the lock will provide greater security than using solenoids and relays because it will eliminate the possibility of a magnetic attack.

Introduction

The goal of this project was to design and create an electronic locking mechanism to provide a medium to low level of security for personnel access control at a reasonable price. The main parts of the project consisted of creating the user interface for the lock and the control circuitry for the lock. The user interface consists of a 3x4 numeric keypad, an ID12LA radio-frequency identification reader, and a 1602 liquid-crystal display with backlight. The control circuitry consists of an Arduino Nano microcontroller and a NEMA-17 Stepper motor. The separate Arduino control and user interface circuits were used with a stepper motor because they can provide a higher level of security compared to the use of relays and solenoids, which can be easily activated by unauthorized users using a magnet (LockPickingLawyer, n.d.). By separating the control circuitry from the user interface, the Arduino can be placed on the inside of the door, thus greatly reducing the vulnerability to a magnetic attack by using the walls of a building as insulation. The stepper motor provides greater security than a solenoid activated lock because it requires a very specifically controlled magnetic field to turn the rotor rather than simply the presence of a magnetic field. The budget goal for the project was to remain under $200, which would put the development of the lock in the same range as the commercial price of similar locks (The Home Depot, n.d.).

Hardware Design

Circuit Design

As the majority of the complexity of this project lies in the Arduino code, the hardware design is relatively simple. The main part of the circuit is the Arduino Nano microcontroller. The inputs to the Arduino are a 3x4 matrix keypad, an ID12LA RFID reader module, a pushbutton switch which is used as an unlock button on the secure side of the door, and a limit switch that would be mounted to the doorframe that is used to reset the lock when door is closed. The outputs of the circuit are a 1602 liquid-crystal display used for prompting the user and displaying the state of the lock and a BIQU A4988 stepper motor driver which drives a NEMA-17 stepper motor. The circuit is powered by a 12VDC power supply with a 9VDC battery backup system. Several resistors are used in the circuit for current and voltage limiting purposes for the LCD contrast setting, the switch inputs, and the battery backup circuit.

The battery backup circuit was adapted from a project found on the website All About Circuits (Smith, 2016). In the event of a main power failure, the device will automatically switch to the battery backup power. The battery being used for this project has a capacity of 600mAh, so a 2kΩ resistor was chosen for the battery backup circuit, which creates a current charging the battery of 1.5mA (12v-9v = 3v, 3v/2kΩ = 1.5mA). Based on the information found on All About Circuits, this charge rate should be sufficiently small enough to safely use the 600mAh battery as a backup. The project consumes approximately 3A of current while turning the motor, meaning that using a fully charged 600mAh battery allows the project to run for just over 10 minutes. This time should be enough to allow anyone inside the secure area to make their way to the door and press the unlock button to escape.

The resistors in the voltage divider to set the contrast of the LCD were chosen by testing various values between 220Ω and 10kΩ to see what worked best. An acceptable contrast was reached with the values 3.6kΩ and 360Ω.The rest of the circuit design is simply connecting the stepper motor to the driver and then connecting the proper inputs and outputs from the Arduino to the matching pins on the switches, LCD, keypad, RFID reader, and stepper motor driver. The schematic diagram for the project is shown in Figure 1 below.

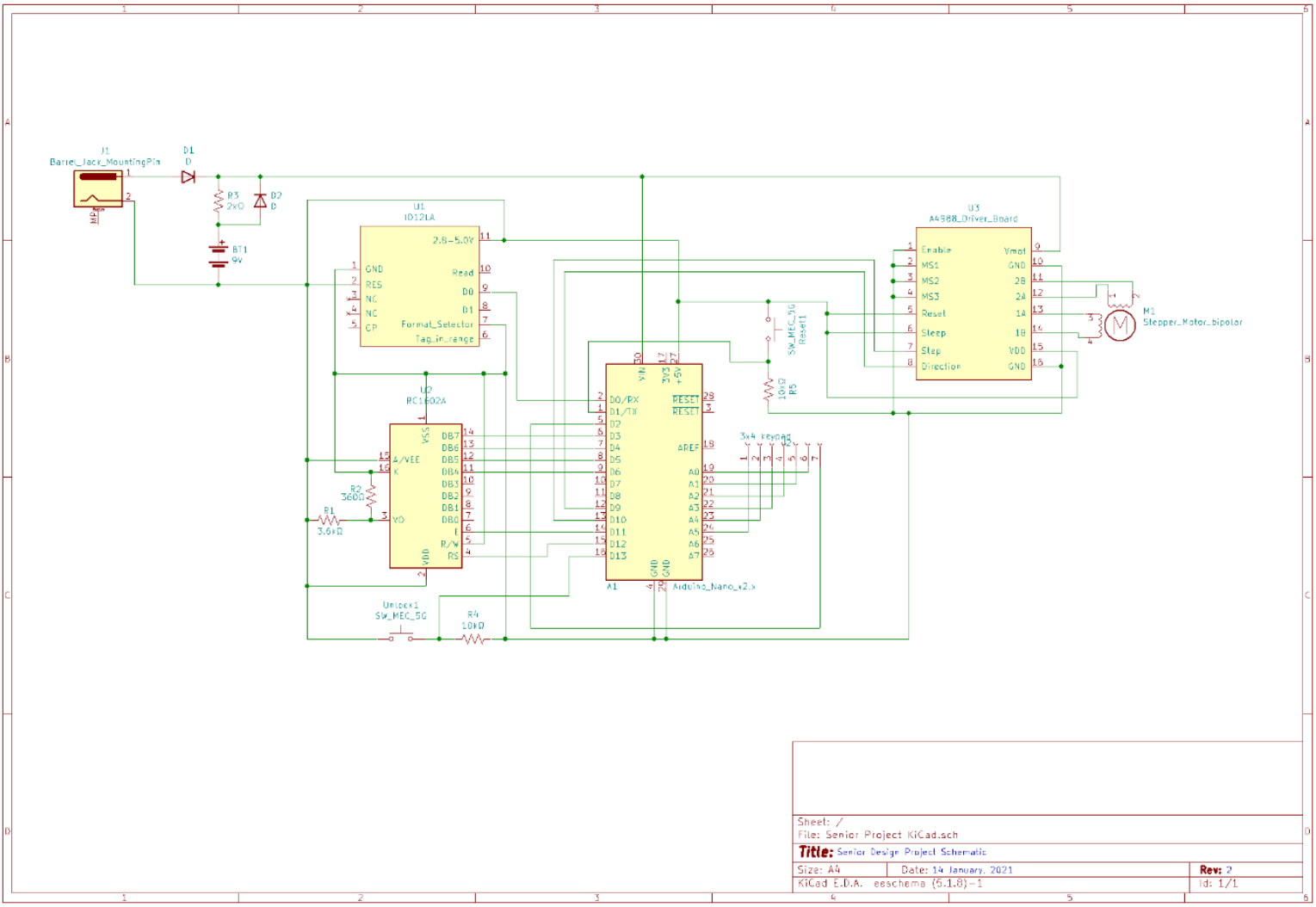

Figure 1

PCB Design

There were two separate PCBs created for this project which were manufactured as one board with breakaway tabs to allow for easy separation after fabrication. One PCB design was solely for the RFID reader, and was a simple rectangular board with mounting holes. The main PCB design had the Arduino, resistors, diodes, and connectors soldered to the board, with pin headers to connect to the other components via cables.

Originally, the stepper motor driver was also going to be soldered to the main board, but it was discovered during testing that the motor and driver pair that were originally going to be used could not provide sufficient torque to turn the spindle of the doorlatch. Because of this, a new motor (the NEMA-17) and driver (a BIQU A4988 driver board) were used, and the original driver was replaced with pin headers which allowed for connection to the new driver. This change also required that a jumper be added from the main power supply to the driver headers because the Arduino could not supply sufficient current to drive the motor at the required torque. Images of both PCBs before and after population can be found below.

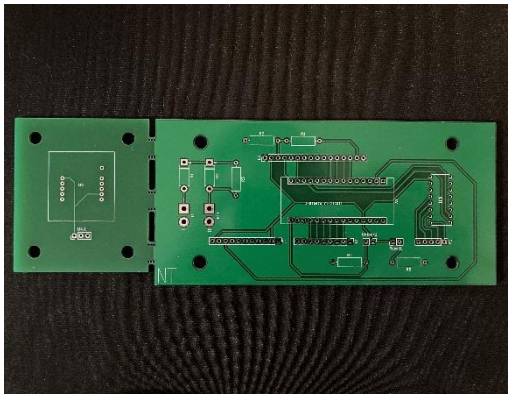

Figure 2

Figure 3

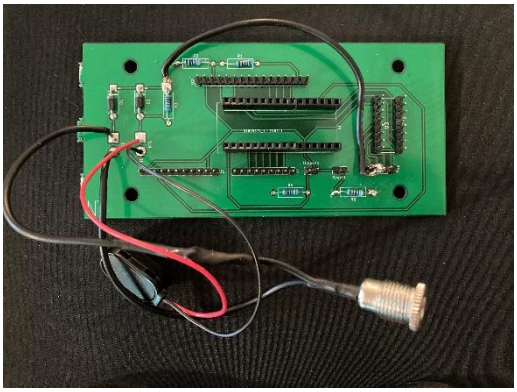

Figure 4

Other Hardware

The housing for the project was two project boxes purchased from Amazon.com. Two separate housings were used to allow for the control circuitry to be located on the inside of the door while the user interface was located on the outside of the door. This seperation of user interface and control circuitry creates a more secure device because no unauthorized personnel would be able to access the control circuitry.

A custom spindle for the doorlatch had to be 3D printed to fit both the stepper motor shaft and the latch mechanism. The spindle was printed in three parts. The first part slid onto the motor shaft and was a 10mm x 10mm x 30mm piece with a hole the size of the motor shaft in the center. The second part went inside the latch mechansim and was 7mm x 7mm x 100mm with a hole slightly smaller than the hole in the first part. The third piece connected the two, and was the same size and shape as the motor shaft; a 14mm long partial cylinder with a diameter of 5mm with one flat side which made the width 4.5mm. This third piece was inserted into the first piece halfway and the filed down to fit inside of the hole in the second piece. This three part design was created because simply using a 7mm x 7mm rectangle did not provide enough clearance on the edges to make a large enough hole for the motor shaft without severely compromising the strength of the spindle. An image of the three spindle pieces is shown on the following page.

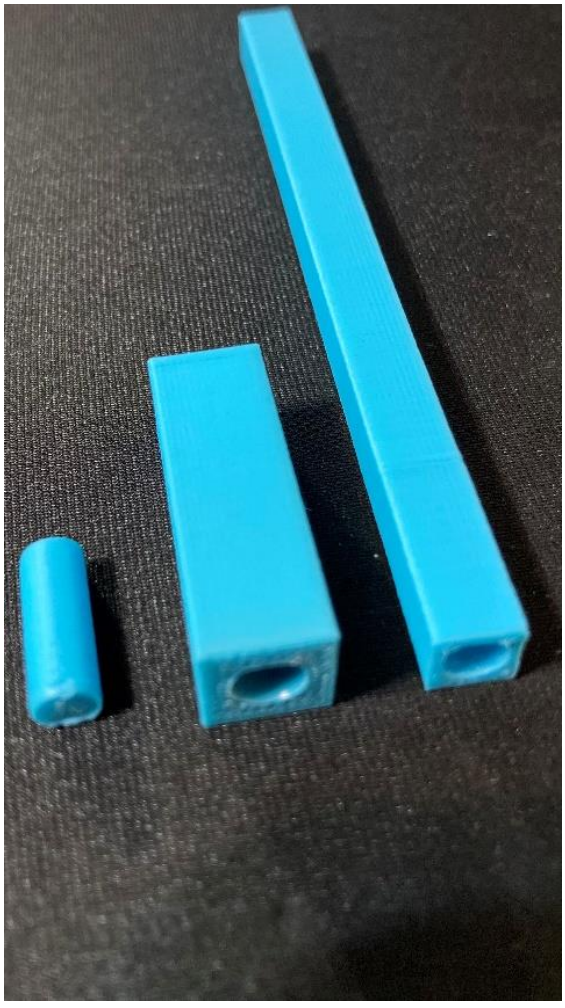

Figure 5

Software Design

Libraries and Set Up

Several libraries were used in the Arduino code for this project, including some non-standard open-source libraries created by GitHub users. The standard libraries used were the LiquidCrystal.h, SoftwareSerial.h, and Stepper.h libraries. The open-source libraries used were Keypad.h library for interfacing with the keypad and the EasyButton.h library for debouncing the unlock and reset switches.

```
 1 /*
 2     Nicholas Tamburrino
 3     Senior Project Arduino Code
 4     Spring 2021
 5     Electronic Door Lock
 6 */
 7
 8 #include <LiquidCrystal.h>
 9 #include <SoftwareSerial.h>
10 #include <Keypad.h>
11 #include <EasyButton.h>
12 #include <Stepper.h>
```

The next lines of code set up constants and create global variables and objects to use in the main part of the code.

```
15 const int RX = 0, TX = 20; //Pins 0 and 20 will be RX and TX
for SoftwareSerial.
16 SoftwareSerial RFID(RX, TX);
17
18 const int RS = 12, EN = 11, D4 = 6, D5 = 5, D6 = 4, D7 = 3;
//Pins for LCD.
19 LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);
20
```

```
21 const int STEPS_PER_SLIDE = 200;       //Number of motor
steps required to slide door latch.
22 const int M1 = 9, M2 = 10;  //Pins to control stepper motor.
23 Stepper motor(STEPS_PER_SLIDE, M1, M2);  //Set up stepper
motor.
24
25 const int ROW_NUM = 4; //Keypad rows.
26 const int COLUMN_NUM = 3; //Keypad columns.
27
28 byte pin_rows[ROW_NUM] = {14, 19, 18, 16}; //Sets up row
pinouts of the keypad.
29 byte pin_column[COLUMN_NUM] = {15, 2, 17}; //Sets up column
pinouts of the keypad.
30
31 char keys[ROW_NUM][COLUMN_NUM] = {    //Keypad layout.
32   {'1', '2', '3'},
33   {'4', '5', '6'},
34   {'7', '8', '9'},
35   {'*', '0', '#'}
36 };
37
38 Keypad keypad = Keypad( makeKeymap(keys), pin_rows,
pin_column, ROW_NUM, COLUMN_NUM );  //Set up keypad.
39
40 const int UNLOCK_BUTTON = 13; //Pin number for unlock
pushbutton.
41 const int RESET = 1;          //Pin number reset limit
switch.
42 const int DEBOUNCE = 100;     //Argument for debounce
perameter of EasyButton.
43
44 EasyButton unlockButton(UNLOCK_BUTTON, DEBOUNCE); //Debounce
unlock pushbutton.
45 EasyButton resetSwitch(RESET, DEBOUNCE);          //Debounce
reset switch.
46
47 const byte UNLOCK[8] =  { //Custom UNLOCK symbol.
48   B00100,
49   B01010,
50   B01010,
51   B00001,
52   B00001,
53   B11111,
54   B11111,
```

```
 55   B11111
 56 };
 57
 58 const byte LOCK[8] =  { //Custom LOCK symbol.
 59   B00100,
 60   B01010,
 61   B01010,
 62   B10001,
 63   B10001,
 64   B11111,
 65   B11111,
 66   B11111
 67 };
 68
 69 char passcode[4] = {'0', '0', '0', '0'}; //Array to store
passcode set by user.
 70
 71 char codeEntered[4];  //Variable to store the code that is
entered by user to unlock the door.
 72
 73 char readTag[10];   //Array of variables to hold data from
read tag.
 74
 75 char key = 00; //Variable to store state of keypad input.
 76
 77 int numOfGoodTags;  //Integer variable to hold number of
authorized tags.
 78
 79 unsigned long unlockTime = 0;  //Variable to store time door
was unlocked
 80
 81 bool unlock = false;  //Variable to control state of locking
mechanism.
 82 bool motorDone = false;  //Variable to store whether door
has been physically unlocked.
 83 bool setUp = true;      //Variable to tell whether the
program is in setUp mode or not.
```

The Setup Function

Once the constants, variables, and objects are ready, the setup function executes. This function executes once when the Arduino is powered on and will not be executed again until the Arduino is reset or loses and regains power. The setup function for this project starts by initializing communication between the Arduino and the RFID module. The switch objects are initialized, and the speed of the motor is set as well. Next the LCD is initialized, a message is printed to the screen indicating that the door will unlock, and the motor is activated to unlock the door. Unlocking the door in the setup function is to ensure that the rest of the setup process will take place without the possibility of locking the user out of the room permanently. The last line of code in the setup function calls the changeCode function, which will allow the user to set the desired passcode.

```
 87 void setup()
 88 {
 89   RFID.begin(9600); //Begin serial communication with RFID
reader.
 90
 91   unlockButton.begin(); //Initialize unlock pushbutton.
 92   resetSwitch.begin();  //Initialize reset switch.
 93
 94   motor.setSpeed(60); //Set speed of motor at 60 rpm.
 95
 96   lcd.begin(16, 2); //Set up 16x2 LCD.
 97
 98   lcd.createChar(0, UNLOCK); //Creates a custom char called
UNLOCK.
 99   lcd.createChar(1, LOCK);   //Creates a custom char called
LOCK.
100
101   lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
102   lcd.write((byte)0); //Print 'unlock' character to LCD
screen.
103   lcd.print(" Door unlocked.");  //Print unlocked message to
```

```
LCD screen.
104   motor.step(STEPS_PER_SLIDE);  //Activate motor to unlock
door.
105
106   changeCode(); //Call changeCode function to set passcode.
107 }
```

The Loop Function

After the setup function is executed, the code begins executing the loop function. This function is a continuous loop and will be executed repeatedly until the Arduino is disconnected from power. The loop function for this project begins by calling the getNumOfGoodTags function, which will allow the user to enter how many RFID tags they wish to set up. Next, the goodTags array is initialized with that number of pointers, and memory is allocated to each pointer in the array to allow for the storage of the unique 10-character code of each RFID tag. The next if statement will only execute when the Arduino is in setup mode, meaning that it will only execute on the first iteration of the loop function. The code inside this if statement will lock the door and end setup mode.

```
110 void loop()
111 {
112   getNumOfGoodTags(); //Call get getNumOfGoodTags function.
113
114   char *goodTags[numOfGoodTags];    //Set up an array of
pointers to store the value of authorized tags.
115   for (int i = 0; i < numOfGoodTags; i++)
116   {
117     goodTags[i] = (char *)malloc( 10 * sizeof(char));
//Alloc memory to pointers in goodTags array.
118   }
119
120   changeTags(goodTags); //Call changeTags function.
121
122   delay(1000);
```

```
123
124   if (setUp)  //If program is in setup mode...
125   {
126     motor.step(-STEPS_PER_SLIDE); //Activate motor to lock
door.
127
128     lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
129     lcd.write((byte)1); //Print 'lock' character to LCD
screen.
130     lcd.print(" Door Locked");  //Print door locked message.
131
132     delay(1000);
133
134     setUp = false;  //Exit setup mode.
135   }
```

Once the set up is complete, the LCD is cleared and a custom character indicating that the door is locked is printed to the screen as well as a prompt to scan an RFID fob or enter the passcode. The Arduino the enters a while loop, which will run until an authorized user selects the option to change the authorized RFID tags. The code in this while loop is the main part of the code for this project. It checks the state of the unlockButton object and sets the unlock variable to 'true' when the button is pushed. If the unlock button is not pushed, the availability of serial data from the RFID reader is checked. If there is data available, the getTag function is called which will read the serial data and store it in an array, then the compareTags function is called which will compare the array created by the getTag function to the goodTags array. If there was no serial data available and the unlock button was not pressed, then the state of the keypad will be read. If a key was pressed, two conditions will be checked. The first condition to be checked is whether the door is unlocked. If the door is unlocked, then the value of the key that was pressed will also be checked. If the value of the key that was pressed while the door was unlocked is '*' the changeCode function will be called to allow the user to set a new passcode. If the value of the key pressed while the door was unlocked is '#' the while loop will be broken, causing the loop

function to end and execute again from the top which will force the goodTags array to be set up again. If the door was not unlocked when a key was pressed, the getCode function will be called to store the value of the key press in an array to later be compared to the passcode. After the value of the key press is used in this block of code, the value of the key variable is reset to null.

```
137   lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
138   lcd.write((byte)1); //Print 'lock' character to LCD
screen.
139   lcd.print(" Scan fob or");  //Print input prompt.
140   lcd.setCursor(0,1);
141   lcd.print("enter code: ");
142
143   while (1) //Loop continuously
144   {
145     if ( !unlockButton.read())  //If unlock button is
pressed...
146     {
147       unlock = true;  //Set unlock variable to true.
148     }
149
150     else if (RFID.available() > 0) //If unlock button was
not pressed and serial data is available...
151     {
152       getTag();
153
154       compareTags(goodTags);
155     }
156
157     else  //If unlock button was not pressed and serial data
was not available...
158     {
159       key = keypad.getKey(); //Read state of keypad.
160
161       if (key) //If key was pressed...
162       {
163         if (unlock == true && key == '*') //If door is
unlocked and '*' key was pressed.
164         {
165           changeCode(); //Call changeCode function.
166         }
```

```
167         else if (unlock == true && key == '#')  //If door is
unlocked and '#' key was pressed.
168         {
169           break;  //End continuous while loop, causing tag
values to be reinitialized.
170         }
171         else getCode(); //Call getCode function.
172
173         key = 00;  //Reset key variable.
174       }
175     }
```

The last part of the while loop checks whether the door should unlock and whether the motor has been activated. If the door should unlock and the motor has not yet been activated, the unlockDoor function is called, which will activate the motor, unlock the door, and record the time at which the door was unlocked. If five seconds have passed since the door was unlocked, the reset switch is pressed, and the motor has been activated, the lockDoor function will be called which will lock the door.

```
177     if ( unlock && !motorDone ) //If door should unlock and
motor has not yet been activated...
178     {
179       unlockDoor(); //Call unlockDoor function.
180     }
181     else if ( (millis() > (unlockTime + 5000)) &&
!resetSwitch.read() && motorDone)  //If door is closed, 5
seconds have passed since door was unlocked, and motor was
actived...
182     {
183       lockDoor(); //Call lockDoor function.
184     }
185   }   //End while loop
186 }  //End loop function
```

The changeCode Function

The changeCode function allows authorized users to change the passcode for the door lock. It begins by setting up a local Boolean variable to use as a flag in the following while loop. The code inside the while loop first clears the LCD and then prints a prompt to enter a new passcode. The keys that are pressed are printed to the screen and stored in the passcode array until it is full. Once four digits have been entered, a confirmation prompt is printed to the screen. The while loop will then either reset and set up the passcode array again or terminate and end the function based on the user's choice.

```
189 void changeCode() //Function to change passcode.
190 {
191   bool done = false;  //Local variable to determine whether
or not to terminate function.
192
193   while (!done) //While done is false...
194   {
195     lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
196     lcd.print("Enter new code:"); //Print new passcode
prompt to LCD screen.
197     lcd.setCursor(0,1);   //Set position of LCD cursor to
first char of second line.
198
199     for (int i = 0; i < 4; i++)
200     {
201       do
202       {
203         key = keypad.getKey(); //Call getKey function.
204       } while (key == 00);  //Loop until new key press is
detected.
205
206       passcode[i] = key;  //Set each digit of passcode to
the value of the button that was pressed on keypad.
207
208       lcd.print(key);   //Print value of key to LCD screen.
209
210       key = 00;  //Reset key.
```

```
211     }
212
213     delay(1000);
214     lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
215     lcd.print("Hit * to accept,");  //Pring confirmation
prompt.
216     lcd.setCursor(0,1);
217     lcd.print("# to reset.");
218
219     do
220     {
221       key = keypad.getKey(); //Call getKey function.
222
223       if (key == '*')
224       {
225         done = true;  //Set done to true if key pressed is
'*'.
226       }
227     } while (key == 00); //Loop until new key press is
detected.
228
229     key = 00;  //Reset key.
230   }
231 }
```

The getCode Function

The getCode function compares the set passcode to the code entered by someone trying to unlock the door. The function sets up a static local integer variable to store the number of digits entered so far by the user, and a local integer variable to store the number of digits entered that match the set passcode. The value of the key pressed by the user is stored in the next available index in the codeEntered array, and the count variable is incremented. Once four digits have been entered, the codeEntered array is compared to the passcode array. If any digits do not

match, the comparison will stop, and the function will terminate. If all four digits match, the unlock variable will be set to true before the function terminates.

```
234 void getCode()
235 {
236   static int count = 0; //Static local variable to store
number of keys pressed.
237
238   int goodDigits = 0; //Local variable to store number good
digits detected.
239
240   codeEntered[count] = key; //Store the value entered for
key in the proper index of codeEntered.
241
242   lcd.print('*');
243
244   count++;  //Increment count.
245
246   if (count == 4)
247   {
248     delay(500);
249
250     count = 0;  //Reset count once codeEntered has all 4
digits.
251
252     for (int i = 0; i < 4; i++) //Loop to compare
codeEntered to passcode.
253     {
254       if (codeEntered[i] == passcode[i])  //If codeEntered
matches...
255       {
256         goodDigits++; //count the number of matching digits.
257       }
258       else  //If codeEntered does not match...
259       {
260         lcd.setCursor(11,1);     //Clear *'s from LCD screen
and set cursor at 11,1.
261         lcd.print("     ");
262         lcd.setCursor(11,1);
263
264         break; //Stop comparing if a digit does not match.
265       }
```

```
266     }
267
268     if (goodDigits == 4)  //If all 4 digits matched...
269     {
270       unlock = true;  //Set unlock variable to true.
271     }
272   }
273 }
```

The changeTags Function

The changeTags function accepts an array of character pointers as an argument and sets up the tags that will be for authorized users. The first while loop clears the serial buffer to ensure that the data read is good. The subsequent for loop will iterate once for each good tag that is to be set up. The loop will prompt the user to scan a tag and wait for them to do so. Once a tag is scanned, the data is checked for the Start of Text character. If it is detected, data that was read will be stored in the first row of the tagsToSetUp array. When the goodTags array is passed by reference to this function, the goodTags array will also be updated with this data. Once the tag data is successfully stored in the array, extra characters are removed from the serial buffer and a confirmation message is printed to the screen.

```
276 void changeTags(char *tagsToSetUp[])  //Function to change
goodTag.
277 {
278   while (RFID.available() > 0)
279   {
280     RFID.read();  //Clear buffer to ensure a good read.
281   }
282
283   for (int j = 0; j < numOfGoodTags; j++ )
284   {
285     lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
286     lcd.print("Scan tag #");  //Print scan tag prompt to LCD
```

```
screen.
287     lcd.print(j+1);
288
289     while (RFID.available() == 0);  //Wait for serial data
before continuing
290
291     delay(20);  //Wait to ensure all data arrives in buffer
before reading.
292
293     if (RFID.peek() == 2) //If Start of Text char is
detected...
294     {
295       RFID.read();  //Clear Start of Text character from the
buffer.
296
297       for (int i = 0; i < 10; i++)  //Read the ten
characters of tag data.
298       {
299         tagsToSetUp[j][i] = RFID.read();
300       }
301
302       RFID.read();  //Clear first checksum from buffer.
303       RFID.read();  //Clear second checksum from buffer.
304       RFID.read();  //Clear a carriage return from the
buffer.
305       RFID.read();  //Clear a line feed character form the
buffer.
306
307       while (RFID.available() > 0)  //Wait for data to
become unavailable.
308       {
309         RFID.read();
310       }
311     }
312     else  //Start of Text char was not detected.
313     {
314       while (RFID.available() > 0)
315       {
316         RFID.read();  //Clear buffer.
317       }
318     }
319
320     lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
```

```
321      lcd.print("New tag accepted");  //Print scan tag prompt
to LCD screen.
322
323      delay(1000);
324    }
325 }
```

The getNumOfGoodTags Function

The getNumOfGoodTags function gets the number of desired authorized RFID tags from the user. The maximum number of authorized tags is 99. A local array of characters is used to store the keypad input from the user, a local integer variable is used to count the number of loop iterations, and a local Boolean variable is used as a flag to determine whether a loop should continue. A prompt for the user to enter the desired number of tags is printed to the LCD, and a while loop begins. Inside the while loop, the state of the keypad is read, and if a number is pressed, the value is stored in the least significant digit of the numberOfTags array. If a second number is entered, the value is stored in the most significant digit of the numberOfTags array. If the key pressed is '*' or '#', the loop will either terminate or the numberOfTags array and counter variable will be reset to begin again. Once the loop is terminated, the characters in the numberOfTags array are converted to a single string variable with the digits in their proper places. This string variable is then converted to an integer and stored in the global numOfGoodTags variable and a confirmation message is printed to the LCD screen.

```
328 void getNumOfGoodTags()
329 {
330   char numberOfTags[] = "xx";  //Array of char variables to
hold number of authorized tags.
331
332   int counter = 0;  //Variable to count loop iterations.
333
```

```
334   bool setUp = true;  //Flag variable to determine whether
set up loop should continue.
335
336   lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
337   lcd.print("Enter desired #");  //Print input prompt to LCD
screen.
338   lcd.setCursor(0,1);
339   lcd.print("of allowed tags.");
340
341   while (setUp)  //Set up loop.
342   {
343     key = keypad.getKey();
344
345     if ( (key != '*') && (key != '#') && (key != 00)  )
346     {
347       numberOfTags[counter] = key;  //Store key in the least
significant open digit in numberOfTags[].
348
349       counter++;  //Decrement counter.
350
351       lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
352       lcd.print("Hit * to accept");  //Pring confirmation
prompt.
353       lcd.setCursor(0,1);
354       lcd.print("# to reset.");
355
356     }
357     else if ( key == '*' )
358     {
359       setUp = false; //End set up loop.
360     }
361     else if ( key == '#' )
362     {
363       counter = 0; //Reset counter.
364       numberOfTags[0] = 'x';
365       numberOfTags[1] = 'x';
366       lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
367       lcd.print("Enter desired #");  //Print input prompt to
LCD screen.
368       lcd.setCursor(0,1);
369       lcd.print("of allowed tags.");
```

```
370     }
371
372     key = 00;   //Reset key.
373   }
374
375   String numOfTagString = ""; //Create empty string to store
number of tags.
376
377   if (numberOfTags[1] == 'x') //If a single digit number was
entered...
378   {
379     numOfTagString = numberOfTags[0]; //Store desired number
of tags in numOfTagString.
380   }
381   else
382   {
383     for (int i = 0; i < 2; i++)
384     {
385       numOfTagString = numOfTagString + numberOfTags[i];
//Store desired number of tags in numOfTagString.
386     }
387   }
388
389   numOfGoodTags = numOfTagString.toInt();  //Convert string
to integer and set number of authorized tags.
390
391   lcd.clear();
392   lcd.print(numOfGoodTags);
393   lcd.print(" tags will");
394   lcd.setCursor(0,1);
395   lcd.print("be set up.");
396
397   delay(1000);
398 }
```

The getTag Function

The getTag function reads data from the RFID reader and stores it in an array to be compared to the goodTags array later. The reading process in this function is the same as the

process in the changeTags function, but only one 10-character tag is read, and its value is stored in the readTag array.

```
401 void getTag()  //Function to read RFID input.
402 {
403   delay(20);  //Wait for 20ms to ensure all data arrives in
serial buffer before reading.
404
405   if (RFID.peek() == 2) //If Start of Text char is
detected...
406   {
407     RFID.read();  //Clear Start of Text character from the
buffer.
408
409     for (int i = 0; i < 10; i++)  //Read the ten characters
of tag data.
410     {
411       readTag[i] = RFID.read();
412     }
413
414     RFID.read();  //Clear first checksum from buffer.
415     RFID.read();  //Clear second checksum from buffer.
416     RFID.read();  //Clear a carriage return from the buffer.
417     RFID.read();  //Clear a line feed character form the
buffer.
418
419     while (RFID.available() > 0)
420     {
421       RFID.read();  //Clear buffer.
422     }
423   }
424   else  //Start of Text char was not detected.
425   {
426     while (RFID.available() > 0)
427     {
428       RFID.read();  //Clear buffer.
429     }
430   }
431 }
```

The compareTags Function

The compareTags function accepts an array of character pointers as an argument and compares the value of the tags stored in the passed array to the value stored in the goodTag array. The function uses a Boolean variable as a flag to determine whether the goodTag array matches any of the tags stored in the tagsToCompareTo array. If any of the tags in the tagsToCompareTo array match the tag in the goodTag array, the unlock variable is set to true. Otherwise, the function terminates without changing anything.

```
434 void compareTags(char *tagsToCompareTo[])
435 {
436   bool tagsMatch; //Variable to store whether readTag
matches an authorized tag.
437
438   for (int j = 0; j < numOfGoodTags; j++) //Outer loop to
compare readTag to authorized tags.
439   {
440     tagsMatch = true; //Reset tagsMatch.
441
442     for (int i = 0; i < 10; i++)  //Inner loop to compare
readTag to authorized tags.
443     {
444       if ( readTag[i] != tagsToCompareTo[j][i] ) //If chars
do not match...
445       {
446         tagsMatch = false;  //Update match.
447         break;  //End inner for loop.
448       }
449     }
450
451     if (tagsMatch)  //If readTag matched an authorized
tag...
452     {
453       unlock = true;  //Set unlock variable to true.
454       break;  //End outer for loop.
455     }
456   }
457 }
```

The unlockDoor Function

The unlockDoor function prints a message to the LCD to indicate the door will unlock and then activates the motor to unlock the door. The global motorDone variable is then set to 'true' and the value of the millis function, which counts the milliseconds since the program began executing, is recorded in the global unlockTime variable.

```
460 void unlockDoor() //Function to control locking mechanism.
461 {
462   lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
463   lcd.write((byte)0); //Print 'unlock' character to LCD
screen.
464   lcd.print(" Door unlocked.");  //Print unlocked message to
LCD screen.
467
468   motor.step(STEPS_PER_SLIDE);  //Activate motor to unlock
door.
469
470   motorDone = true;  //Update motorDone.
471
472   unlockTime = millis();  //Set new value for unlockTime.
473 }
```

The lockDoor Function

The lockDoor function activates the motor to lock the door, resets the global unlock and motorDone variables, and prints a prompt for the user to scan a fob or enter the code to the LCD.

```
476 void lockDoor() //Function to lock door.
477 {
478   motor.step(-STEPS_PER_SLIDE); //Activate motor to lock
door.
479
480   unlock = false; //Reset unlock condition.
481
482   motorDone = false;  //Reset motorDone condition.
```

```
483
484   lcd.clear();  //Clear LCD screen and set cursor to top
left corner.
485   lcd.write((byte)1); //Print 'lock' character to LCD
screen.
486   lcd.print(" Scan fob or");  //Print input prompt.
487   lcd.setCursor(0,1);
488   lcd.print("enter code: ");
489 }
```

Budget and Parts List

The goal for the project was to keep the total cost under $200. This goal was accomplished with the total cost amounting to just under $130. This price puts the development cost of the project in a very similar price range to the retail price of similar electronic locks. A list of parts along with their estimated and actual costs can be found below.

ITEM	Estimated Cost	Actual Cost
ID12LA RFID Reader	$30.00	$29.95
Arduino Nano	$20.00	$18.63
16x2 LCD	$10.00	$3.25
3x4 Matrix Keypad	$3.00	$4.50
Stepper Motor and Driver	$5.00	$16.19
PCBs	$60.00	$5.74
Misc. Resistors (5 @ $0.008 each)	$5.00	$0.04
Housing	$30.00	$19.99
Locking Mechanism	$20.00	$14.99
Pushbutton	$2.00	$1.13
Battery	$3.00	$4.50
Limit Switch	$2.00	$0.50
Diodes	$1.00	$0.60
Power Supply	$6.00	$5.95
Barrel Jack Connector	$2.00	$1.25
Total	$199.00	$127.21

Results

The result of following the methodology and implementing the software described above was a fully functioning electronic door lock. The Arduino control circuit and user interface were placed in the two separate housings to allow for their proper separation, and the stepper motor, spindle, and door latch were assembled and tested. However, on the first test run of the circuit, two problems were discovered. First, the stepper motor that was being used could not provide sufficient torque to turn the spindle while in the door latch, and second, the LCD was not displaying any characters.

The first issue was resolved by purchasing a more powerful stepper motor (a NEMA-17) and driver (a BIQU A4988 driver board). In order to accommodate this change, the old driver was removed from the PCB and a jumper wire was added, connecting the positive supply voltage to the pin headers that were previously used to connect to the motor. These adjustments to the PCB allowed for new pin headers to be soldered to the board which could be used to connect to the new motor and driver.

The second issue was resolved by removing the LCD from the project and testing it independently. The independent test of the LCD yielded the same results as the project test, with the LCD showing no characters. A second LCD was tested using the same circuit as the previous independent test, and it resulted in the LCD working as expected. From this result, it was determined that the first LCD was somehow damaged and was deemed inoperable. The second LCD was then tested in the project, but again no characters were displayed on the screen. Another independent test of the second LCD was conducted, this time with the same resistor divider being used to set the contrast as is present in the project. This test confirmed that the

resistor divider in the circuit needed to be adjusted, and through trial and error, new values for the resistor divider were selected.

Once the two issues described above were resolved, the final circuit worked as intended, with the user interface accepting input data and passing it to the Arduino, and the Arduino in turn writing to the LCD and activating the motor when appropriate. Images of the control circuit and user interface in their housings and of the motor and latch can be found below.

Figure 6

Figure 7

Figure 8

Conclusion

Overall, the project was a success. The goal of designing and creating an electronic locking mechanism which could be used for personnel access control was achieved, as well as the goal of remaining in a budget under $200, with the total cost of development reaching just under $130. While there were some minor issues in the development of the project, there were no major delays or obstacles that degraded the quality of the final product. The results of this project show that it is possible to create an electronic locking mechanism that has a greater level of security than a relay or solenoid controlled mechanism without greatly increasing the price of such devices.

References

LockPickingLawyer. (n.d.) *Home* [YouTube Channel]. https://www.youtube.com/c/lockpickinglawyer/featured

The Home Depot. (n.d.). Retrieved April 12, 2021, from https://www.homedepot.com/b/Hardware-Door-Hardware-Door-Locks-Keyless-DoorLocks/N-5yc1vZc2bdZ12l3/Ntk-EnrichedProductInfo/Ntt-electronic%2Bdoor%2Block?NCNI-5&sortby=bestmatch&sortorder=none&storeSelection=

Smith, J. P. (2016, February 22). Create your own battery backup power supplies - projects. Retrieved from https://www.allaboutcircuits.com/projects/battery-backup-power-supplies/