SYSC 3010 Group T1

Final Report - Safe Access Security System

Date of Submission: December 6, 2019
Carleton University
1125 Colonel By Drive

Professor: Cheryl Schramm

Github - **https://github.com/kc-carleton/sysc3010-t1**

Group members

Michael Pruss	101008219
Michael Skalecki	100969837
Philip Naida	101012663
Keyan Cassis	101011524

1.0 Project Description

1.1 Problem Motivation

Data confidentiality and security is a growing concern in the modern world. Many institutions and individuals have highly sensitive information that should remain confidential and not be accessed by unlawful individuals. Currently there are many physical unsecure systems used to store confidential information that allow unregistered users access [1]. Such systems pose risks of stolen or leaked information that can be used to damage, bribe, and blackmail owners. Securely storing physical documents and objects is paramount in ensuring their safekeeping and preventing unwanted access [2].

The motivation of the project was to create a security system for multiple users to safely store physical confidential information. The outcome of this project allowed users to store and retrieve confidential and personal documents from multiple safes by entering their unique credentials. The security system prevented unwanted access to secure safes and malicious access to private documents by unauthorized individuals.

1.2 Problem Statement

The safe access system is a robust security system that allows users to store physical confidential documents in multiple safes for easy and secure access. The safe access system will be administered by Safety Security Inc. that will control user code creations, safe user allocation, and will monitor malicious access. This system has similarities to a bank where users deposit confidential items in physical safes that are managed by Safety Security Inc. employees.

To facilitate user access, users will be able to enter configured user codes, passcodes, and safe numbers to open specific safes which they have been granted access to by Safety Security Inc. Multiple users may be permitted to access the same safe with their own individual user_code/passcode combinations. Safety Security Inc. employees will distribute safe credentials on a user basis through an administrative android app. Once the user has been given access to a safe, the user will be given a unique user code and passcode through the mobile android app. Using the given user_code, passcode, and the safe number, the user will be able to access their granted secure safe. All of the credentials will be securely saved in a database managed by Safety Security Inc. To ensure paramount security, the passcodes will be hashed upon creation and will be stored as hashed values in the database.

Users will enter their user_code, passcode, and granted safe number to a centralized terminal which will have control of opening all safes within Safety Security Inc through the centralized server. To ensure security, each user’s passcode is hashed and stored in the terminal before being sent to the centralized server. The centralized server will have access to all operable safes and will receive requests to open safes from terminals. Upon receiving a request to open a safe from a terminal, the centralized server will receive the entered user code, the hashed passcode, and the safe number to open. The centralized server will access the database to ensure that the user entered credentials match with the content in the database. Upon successfully matching the credentials, the centralized server will open the specific safe, assuming it has not been already open. The centralized server will send an acknowledgement with a success or failure code to the terminal depending on whether the user credentials matched the database and whether the safe is already opened. Upon entering an incorrect user code, passcode, and safe number pair three times in a row, the centralized server will notify Safety Security Inc. employees through the administrative android app.

2.0 System Architecture Design

2.1 Deployment Diagram

As shown in **Figure 2.1.1** below, the system is separated into four main components that include the Keypad Interface, Central Control, Door Interface, and Android App. The Keypad Interface is responsible for receiving input from the user and displaying safe unlock results to the user. The Keypad Interface also interacts with the Central Control to pass user credentials and receive safe unlock status. The Central Control interfaces with the Door Interface, Database, and a Monitor. The Central Control verifies user credentials with the Database and sends an unlock signal to the Door Interface to unlock the door. The Central Control outputs logs to the Monitor that detail user access. The Door Interface handles the unlocking and locking of the safe. The Door Interface interacts with the DoorMotor and DoorSensor to detect if the safe is open and to unlock and unlock the safe. Due to hardware limitations, only a single Door Interface is used, however the system supports for multiple safes. The Android App allows users to create credentials for safes. The Android App saves the credentials to the Database which the Central Control has access to.

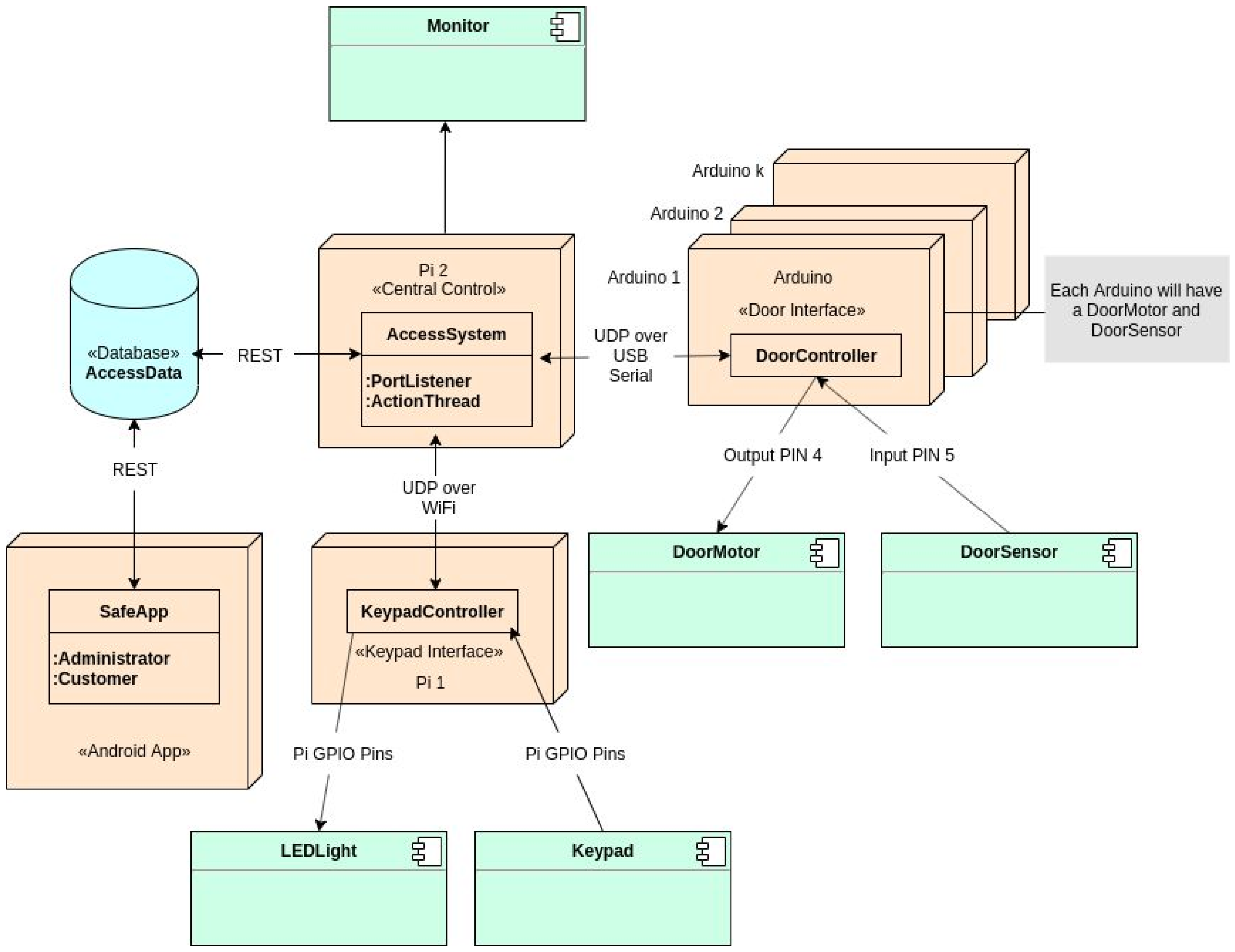

Figure 2.1.1: Deployment Diagram for Safe Access Security System

2.2 Sequence Diagrams

Referring to **Figure 2.2.1** below, the user is prompted to enter a user ID and passcode in order to unlock access to a door. The KeypadController on Pi 1 encrypts the login and sends it to the AccessSystem on Pi 2 over wireless UDP. Pi 2 is in headless mode. The AccessSystem accesses AccessData through REST to see if the login is registered. If the login is registered then the AccessSystem notifies the KeypadController to turn the LED Light to green, and the AccessSystem notifies the DoorController through wired UDP to unlock the door. The DoorController responds once the door is opened and the AccessSystem notifies the KeypadController to turn the LEDLight to red. If the login is not registered then the AccessSystem notifies the KeypadController to turn the LEDLight to red. Administrators can add and remove logins through the AdministrationApp which updates the AccessData using REST. The SafeApp displays the users are registered and the app alerts the administrator if a user fails to type in a correct passcode three times.

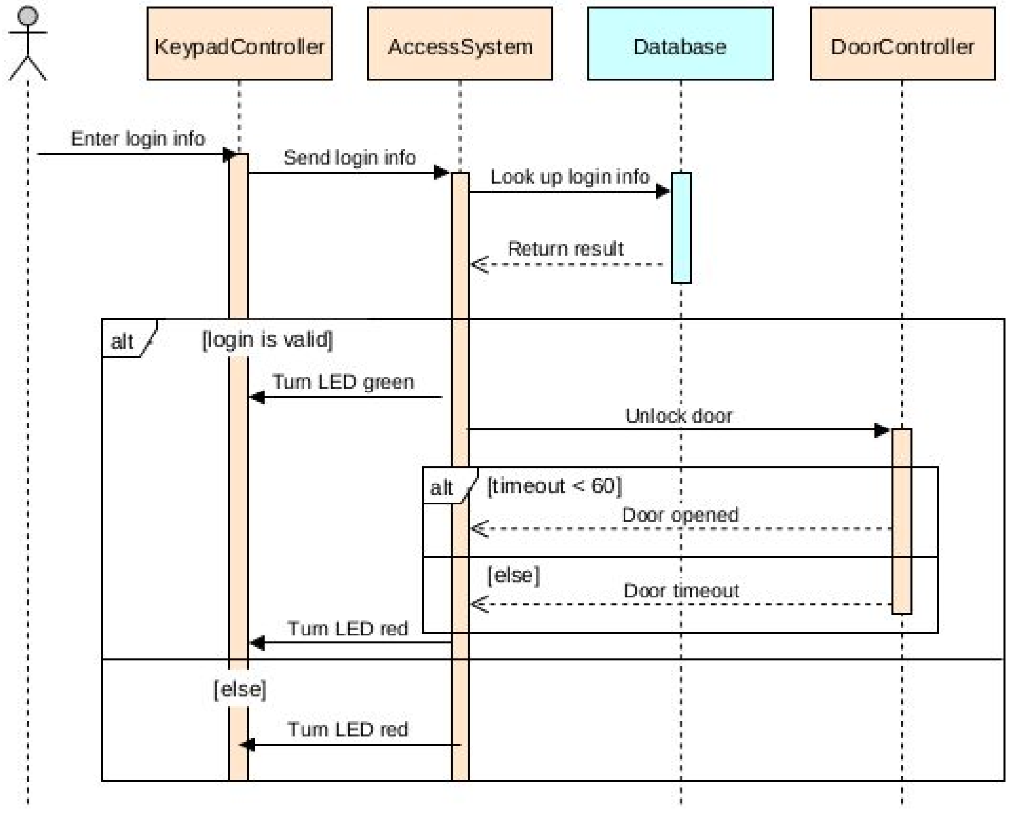

Figure 2.2.1: Sequence Diagram

As shown in **Figure 2.2.1**, the design ensures security by separating user keypad interactions and safe door interface while encrypting all communications between modules. Attackers would not be able to retrieve user Id and password pairs since they are encrypted in the database, in

UDP connection packets, and within the keypad controller. The safe door is also secured by the Door Interface that is only accessed by the Central Control on entering correct user Id and password pairs. The proposed security system will allow individuals to safely store confidential documents within safes and securely retrieve the documents through encrypted passwords and user Ids.

The following sequence diagram in **Figure 2.2.2** shows how the system would respond to a user trying to unlock the safe. In this diagram there are two situations shown which detail what would happen if the user inputs valid credentials, and the second showing if they were invalid credentials.

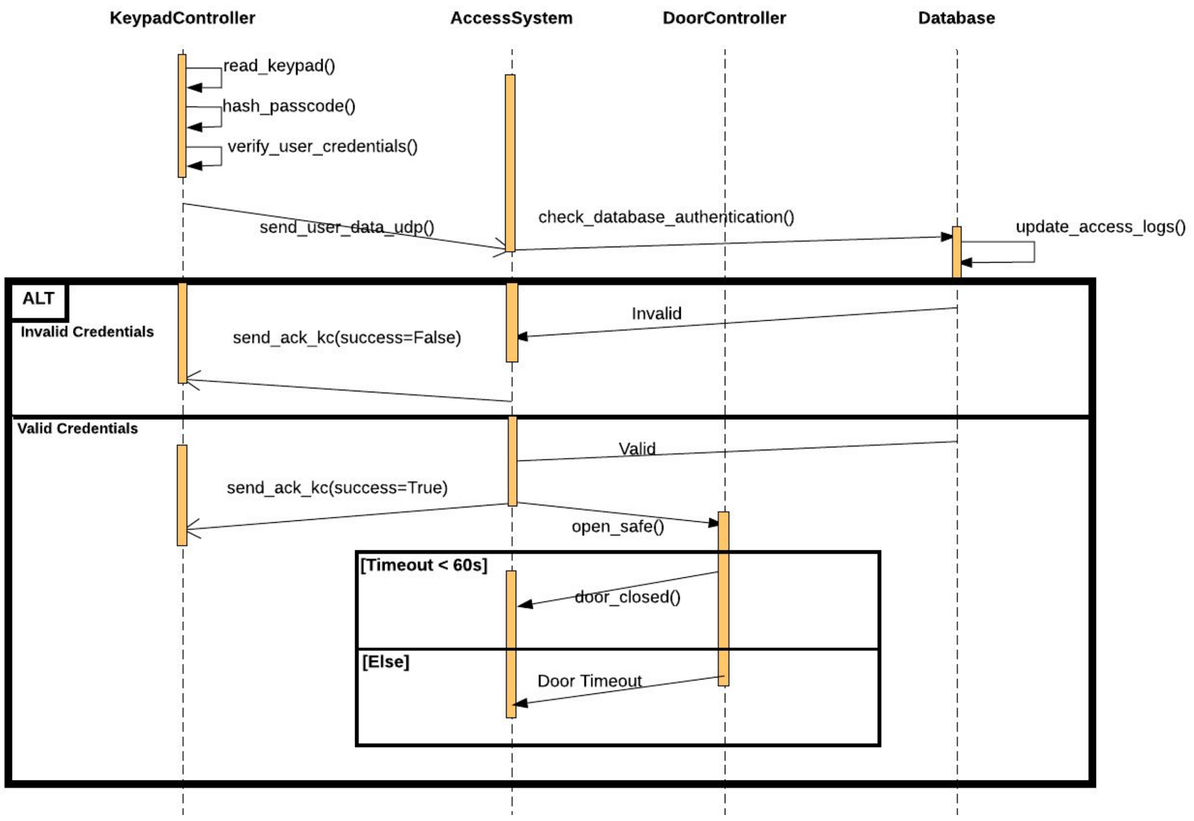

Figure 2.2.2 Detailed Sequence Diagram of System

2.3 Message Protocol Table

The protocol used for UDP communication consisted of an opcode of fixed length, and then based on the opcode type, either data, a command, or an error string followed the opcode finally being terminated with a null character. **Table 2.3.1** shows the 4 different opcodes allowed. **Table 2.3.2 and 2.3.3** show the communication protocol between the Keypad Controller (KC) and the Access System (AS) and the protocol between the AS and the Door Controller (DC) respectively.

Table 2.3.1: Packet Types

Packet Type	Purpose	Opcode Identifier
Data	Data transfer	0x01
Acknowledge	Acknowledgement of received packets	0x02
Command	Issuing commands	0x03
Error	Error notification	0x04

Shown below in **table 2.3.2** is the communication protocol between the KC and the AS. Data packets are identified by the correct data opcode 0x01 and then followed by the json string to be sent and terminated with a null character. Error packets are identified with the correct error opcode 0x04, followed by the error string and terminated with a null character. Acknowledgement packets are identified by the opcode 0x02 and followed by a boolean value. In communication between the KC and the AS, the only acknowledgement packets which will be sent are acknowledging data (containing user credentials) sent from the KC to the AS. The boolean value returned in the acknowledgement packet from the AS to the KC tells the KC whether the credentials entered by the user were correct or not, indicating whether to set the green or red LED. This acknowledgement packet is also terminated with a null character.

Table 2.3.2: Communication Protocol between KC and AS

Sender	Receiver	Packet Type	Packet Format
KC	AS	Data	[0x01, JsonString, ‘\0’]
AS	KC	Error	[0x04, errorMessage, ‘\0’]
AS	KC	Acknowledgement	[0x02, boolean, ‘\0’]

In **table 2.3.3**, the communication protocol between the AS and the DC is outlined. Error packets are identified by the opcode 0x04 and are followed with an error string which is terminated with a null character. Command packets are identified by the opcode 0x03, then follows a command string and a null character. Acknowledgement packets are identified with the opcode 0x02. Following the opcode, the packet contains either True or False. The packet will contain True if the safe was previously locked and the command opened it, or if the safe was already opened when the open command packet was sent, then the acknowledgement packet will contain the value False. Acknowledgment packets are also terminated with a null character.

Table 2.3.3: Communication Protocol between AS and DC

Sender	**Receiver**	**Packet Type**	**Packet Format**
AS	DC	Error	[0x04, errorMessage, ‘\0’]
AS	DC	Command	[0x03, doorAction:boolean, ‘\0’]
AS/DC	AS/DC	Acknowledgement	[0x02, boolean, ‘\0’]

3.0 Discussion of Results

3.1 Changes From Proposal

The general architecture of the system has not changed significantly from the proposal. Everything from the Keypad Interface, Central Control, Door Controller, Database remained. The keypad and door sensor also remained, as well as the LED light and door motor. The Android application was modified to be controlled solely by an administrator rather than having both normal users and administrator users. The safe access logs were moved to be displayed on a monitor rather than within the application. A monitor was added to the AccessSystem to view logs of accessed users.

A significant change from the proposal was the introduction of multiple arduino boards. Each board corresponds to a different safe and controls the unlocking and locking of the safe. Due to the hardware limitations of the project, only a single arduino was used to control a single safe. However, the Central Control, Keypad Controller, Android app, and the database support multiple safes. Instead of using encryption to securely transfer passcodes, hashing was used to hash passcodes prior to them being sent. The hashed values are then compared between the entered user credentials and database of valid credentials. Hashing was introduced because hashing is simpler to implement than private key encryption between the Door Interface and Android App. WiFi communications are also encrypted by default so there is no major security risk in this decision. All components were completed on time and integrated fully before the project deadline.

Slight changes were also made to the DoorController's peripheral circuits. Appendix B shows the proposed arduino circuits from the design report. For the solenoid deadbolt circuit, it was determined that a single 9V battery was not sufficient to power the solenoid in order for it to retract the deadbolt. For this reason, two 9V batteries were placed in series to provide a total voltage of 18V to the solenoid. This resulted in proper and timely functionality of the deadbolt. Regarding the magnetic sensor in Appendix B, it was found that the 220Ω resistor was not necessary as the arduino pin could be programmed to function as a pull-up resistor instead and therefore the resistor was not included in the circuit.

The administrator does not get notified by the Central Control system if a user entered their passcode incorrect three times. Instead, an incorrect passcode counter is incremented by the Central Control in the database. If the counter reaches three, the Android App notifies the admin about the infraction. This was changed to reduce the communications between the Central Control and the Android App to be only through the database.

3.2 Findings

Through development with the Keypad Controller subsystem, limitations were found on the speed of button pressing with the keypad itself. The minimum time between key presses is around half of a second. The keypad used for our design is a strictly hardware interfaced keypad which relies on the user to write drivers to use it as an input device to the system. The reason for this key press speed limitation is due to the way the keypad must be polled for a key press. Debouncing is required in obtaining input from the keypad and because of this, it limits the speed at which the keys can be pressed.

In the future, or in a real world deployed system, a higher quality keypad potentially with optimized software driven interfacing would be used to remove this limitation. Higher quality hardware would remove the debouncing problem and optimized drivers would allow extremely low time in between key presses.

Regarding the DoorController subsystem, it was found that while the chosen solenoid deadbolt was very effective in retracting and locking the bolt, it consumed a large amount of power. This was evident by the numerous 9V batteries that were used to depletion during the course of the project's progression. The magnetic sensor was found to be very accurate in its readings. The hardware specifications for the device listed a maximum distance of 150mm for precise readings. Through experimentation, it was noted that the magnetic sensor in fact required a smaller than listed range to provide proper readings when the safe door is closed; perhaps 50-100mm. This was however in no way detrimental to the functionality and security of the safe as the magnets nearly make contact when the safe door is closed and pushed back into the safe's frame.

4.0 Contributions

Table 4.1: Each members code contributions

Author	**Code**
Michael Pruss	● Central Control and Access System ● test_central_access.py (Tests for Access System) ● Helped with udp_utils ● UDP communications from Central Control to Door Controller and Keypad Interface ● Database
Michael Skalecki	● Android Application ● Database
Philip Naida	● Keypad Controller ● udp_utils (UDP communication library) ● test_udp_utils (tests for UDP library)
Keyan Cassis	● Door Controller main loop and functions ● Door Controller UDP communication ● Door Controller sw/hw test functions

Table 4.2: Each members document section contributions

Author	**Section(s)**
Michael Pruss	Section 1.1, 1.2, 2.1, 2.2, 3.1 6.0 Appendix A
Michael Skalecki	Section 2.1, 2.2, 3.1, 6.0 Appendix A Figures 2.1.1, 2.2.1
Philip Naida	Sections 2.3, 3.2 Figure 2.2.2 Tables 2.3.1, 2.3.2, 2.3.3
Keyan Cassis	Section 3.1, 3.2 Figures 5.3, 5.4

5.0 References

[1] “Dangers of Unsecured Data Systems,” 2019. [Online]. Available: https://its-networks.com/dangers-of-unsecured-data-systems/. [Accessed Sept. 29, 2019].

[2] “Start with Security - And Stick with It,” 2019. [Online]. Available: https://www.ftc.gov/news-events/blogs/business-blog/2017/07/start-security-stick-it. [Accessed Sept. 29, 2019].

[3] “Use a Sparkfun COM-14662 12 digit 3*4 matrix keypad with a Raspberry Pi for PIR alarm system,” 2018. [Online]. Available: https://securipi.co.uk/use-a-sparkfun-com-14662-12-digit-34-matrix-keypad-with-a-raspberry-pi-for-pir-alarm-system [Accessed October 27th 2019]

[4] V. Kartha, "LED Blinking using Raspberry Pi - Python Program", *electroSome*, 2019. [Online]. Available: https://electrosome.com/led-blinking-raspberry-pi/. [Accessed: 03- Dec- 2019].

6.0 APPENDIX A

Github Repository URL: https://github.com/kc-carleton/sysc3010-t1

Code Repository:
The code repository is sectioned off for each component of the system. There are separate folders for the Android app, Keypad Interface, Central Control, and Door Controller. The code for the Android app can be found in the android folder, the Keypad Interface in the keypad_controller folder, the Central Control in the central_control folder, and the Door Interface in the arduino folder. Each folder contains the code to run each system as well as testing code to test each component.

Arduino Setup:

1. In the android/DoorController folder of the git repository, find the file DoorController.ino. Upload this file using the Arduino IDE to the Arduino Uno.
2. Setup the Arduino Uno board following the wiring diagram shown in **Figures 5.3 and 5.4**.
3. Plug ethernet cable to ethernet switch that is connected to the Central Control Pi.
4. Door Controller should now lock and unlock safe based on requests from Central Control Pi.

Keypad Controller Pi Setup:

1. Assuming that the keypad and LEDs are connected to the raspberry pi as shown in **Figures 5.1 and 5.2**, the next step would be to power on the Keypad Controller pi by plugging in the power supply to the micro-USB interface on the raspberry pi.
2. Next, we need to connect to the pi to be able to download and start the keypad software. This can be done by either plugging in a network cable to use for SSH between the pi and another computer or you can connect the raspberry pi's mini-HDMI interface to a monitor as well as connect a keyboard and mouse for use with the raspberry pi. **Note:** *If you are using SSH, make sure you know the IP address of the raspberry pi. For this project, we have set a static IP for easy connectivity.*
3. Once you have connected to the Keypad Controller pi, you should clone the repository from the link above.
4. For this project, static IP addresses have been set on all devices for smooth continuous communication between devices. If you are running this on a new raspberry pi, make sure you change the IP addresses and port numbers at the bottom of the 'keypad_interface.py' file in the 'keypad_controller' subdirectory to the IP address and ports you intend to use in the system.
5. Now that everything has been setup, you may start the keypad software by running the command 'python keypad_interface.py' in the 'keypad_controller' subdirectory. This will tell the raspberry pi to start listening for keypad input. Now you can input your credentials

in the keypad and the Keypad Controller pi will send them to the Access System pi for validation. Just be sure that all the other parts of the system are setup as well or the system will not work.

Central Control Pi Setup:

1. Connect the Central Control Pi via ethernet to the switch which is connected to the Door Controller.
2. Connect a monitor, keypad, and mouse to the Central Control Pi.
3. Ensure that the Central Control Pi is connected to the internet so it can communicate with Keypad Controller and the database. Download the repository using command: git clone https://github.com/kc-carleton/sysc3010-t1.git
4. Execute the central control python file while being in the root of the repository. Use the command line to execute: PYTHONPATH=keypad_controller/ python3 central_access/central_access.py
5. Central Control Pi should now be ready to receive request messages from Keypad Controller and to unlock the safe through the Door Controller.

Database, and Android App:

1. The database is always running, so no setup is needed here.
2. Load the Android application by installing the APK file named SafeApp.apk located in /android on our repo, https://github.com/kc-carleton/sysc3010-t1
3. On the main screen of the app, you will be prompted for an admin login.
   Username: admin
   Password: pw
4. The Admin Controls screen provides the admin with the ability to add new users, remove users, add credentials, remove credentials, as well as a view of all of the registered users.
5. Begin by adding a new user and providing their name.
6. Add a credential by using the user’s usercode and a safe number to access.
7. Remember the passcode from the previous step as this will be hashed and unaccessible.
8. Remove credentials by providing usercode and safe number.
9. Remove users by providing usercode.
10. If a user fails to login 3 times, a notification will automatically pop up within the application.

7.0 APPENDIX B

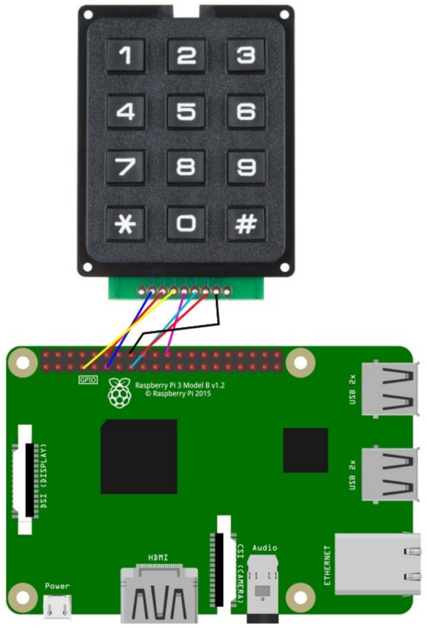

Figure 5.1: Keypad Interface hardware schematic [3]

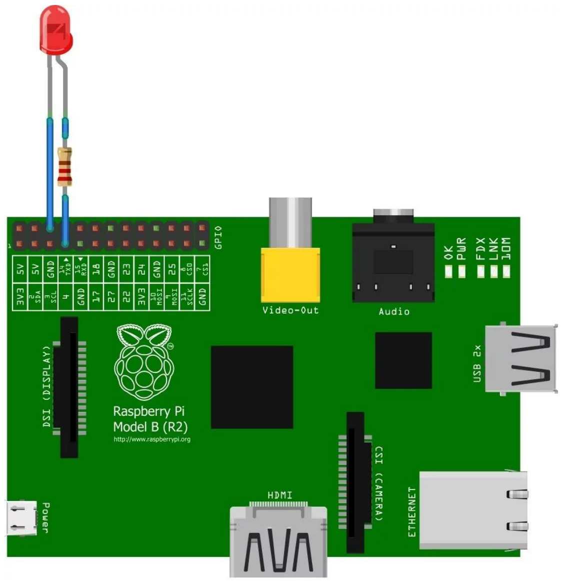

Figure 5.2: LED interface to Pi [4]

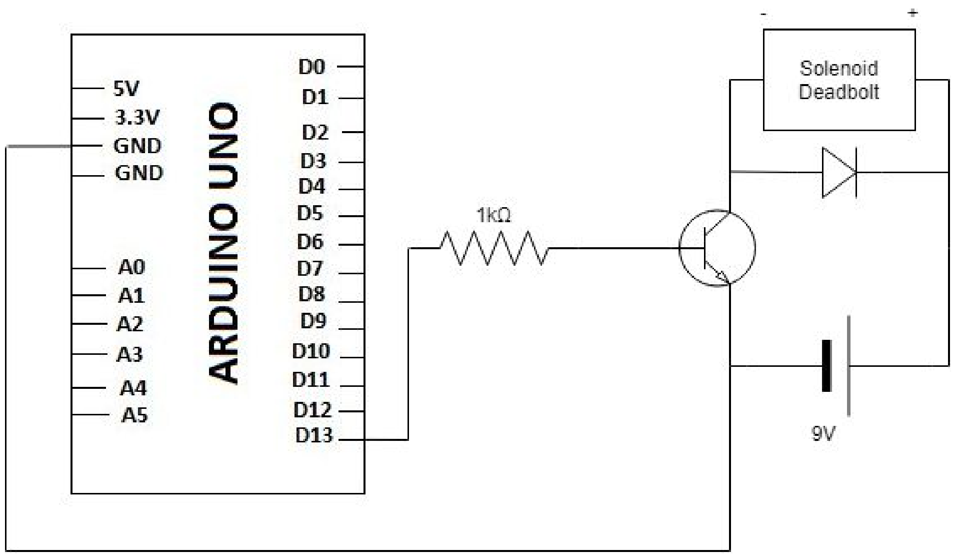

Figure 5.3: Solenoid Deadbolt circuit diagram

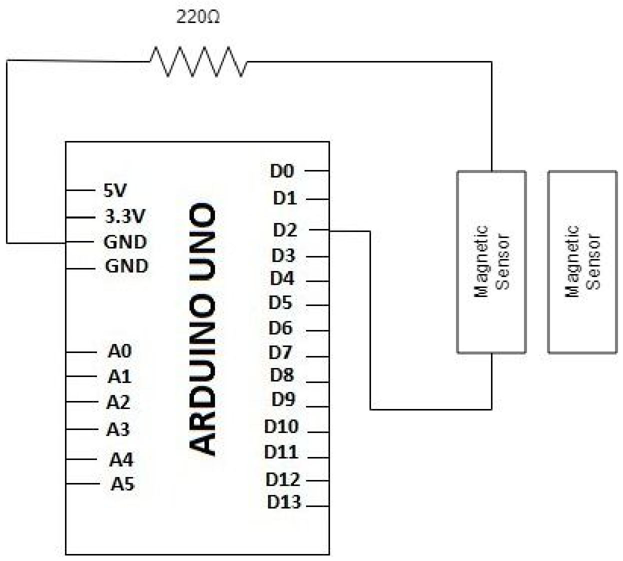

Figure 5.4: Magnetic Sensor circuit diagram