

CMPE123 Senior Design Project: NFC-Enabled Lock Management System

Bowen Brooks, Samuel Wu *

Contact: bojbrook@ucsc.edu, sazwu@ucsc.edu

Abstract

This project is for the Computer Engineering Senior Design Capstone, at University of California, Santa Cruz, CMPE123. The project is meant to create an Internet of Things (Iot) capable smartlock to be used in a building. A room with the cloud-connected smartlock installed will use Near-Field Communication (NFC) technology for easy access. The project emphasizes cloud connectivity, power management, and cost.

*Thanks to Professor Anujan Varma and teaching assistants Sargis Yonan and Sam Mansfield for their guidance during this project

Contents

1	Project Summary (Bowen)	3
2	Figures and Schematic (Sam)	4
2.1	Block Diagram	4
2.2	Overall Software Flow Chart	5
2.3	Circuit Schematic	6
3	Microcontroller (Bowen)	7
3.1	Microcontroller Overview	7
3.2	TI 3220s API	7
3.2.1	StartWiFi()	7
3.2.2	IsAllowed()	7
3.2.3	YellowLight()	7
3.2.4	MotorControl()	8
4	NFC Sensor (Sam)	9
4.1	NFC Sensor Overview	9
4.2	Microcontroller and NFC Sensor interface	9
4.2.1	Command Frame Packet	9
4.2.2	SPI sequence for communication	10
4.3	Serial Peripheral Interface API Calls	11
4.3.1	GetFirmwareVersion()	11
4.3.2	ConfigureSAM()	11
4.3.3	DetectPassiveTarget()	12
4.3.4	InDataExchange()	12
4.4	Sensor and Android phone interface	12
5	Network Structure (Bowen)	14
5.1	Network Overview	14
5.2	Google Compute Engine	14
5.3	Message layout protocols	14
5.4	Google Compute Engine API	14
5.4.1	get_student_classes()	14
5.4.2	get_classes_for_room()	14
5.4.3	login`room()	14
5.4.4	logout`room()	15
5.4.5	change`room`capacity()	15
5.4.6	reset`room`capacity()	15
5.4.7	reset`room`log()	15
6	Google Cloud (Bowen)	15
6.1	Google Cloud Entities (Sam)	16
6.1.1	Room Entity	16
6.1.2	User Entity	16
6.1.3	Room Logs Entity	16

7	Administrative Website (Sam)	16
7.1	Administrative Website Overview	16
7.2	Administration API	17
7.2.1	AddNewUser()	17
7.2.2	AddNewRoom()	17
7.2.3	ModifyRoomEntity()	17
7.2.4	ModifyUserEntity()	17
7.2.5	QueryAllRoomsEntities()	18
7.2.6	QueryAllUserEntities()	18
7.2.7	DeleteRoomEntity()	18
7.2.8	DeleteUserEntity()	18
7.2.9	LogInRoom()	18
7.2.10	LogoutRoom()	18
7.2.11	CountRoomPopulation()	19
7.2.12	ClearRoomLog()	19
7.2.13	HistogramEnterTime():	19
7.2.14	HistogramExitTime():	19
7.2.15	AddClassToRoom()	19
7.2.16	ParseCSVRoomEntity()	20
7.2.17	CSVToCloudRoom()	20
7.2.18	AddClassToUser()	20
7.2.19	ParseCSVUserEntity()	20
7.2.20	CSVToCloudUser()	20
8	Android Application (Bowen)	21
8.1	Android Application Overview	21
8.2	Android Application API Calls	21
8.2.1	GetListOfLabs()	21
8.2.2	GetLabsCapacity()	21
8.2.3	GetRoomLogs()	21
9	Battery (Sam)	21
9.1	Battery Overview	21
9.2	Battery Calculations	22
9.3	Battery Graph	23
10	Schedule (Bowen & Sam)	24
10.1	Winter Quarter	24
10.1.1	Quarter Goals	24
10.1.2	Tasks for the Quarter	24
10.2	Spring Quarter	24
10.2.1	Quarter Goals	24
10.2.2	Tasks for the Quarter	25
11	List of hardware components (Bowen)	26
11.1	Video Link	26

1 Project Summary (Bowen)

This goal of this project is to create a lock management system that could potentially replace the current BSOE door lock system. In rooms with the project's smartlock installed, an administrator will be able to view live statistics and population of each a room, an element not present in the current BSOE infrastructure. Each lock will have a main microcontroller with internet access to Google Cloud and NFC support. A user can access the room by tapping his or her Android smartphone to the lock. Each lock will access Google Cloud Datastore via Google Cloud Compute using TCP/SSL. Additionally, the entire assembly will be relatively low power, with approximately 6 months of battery life off of two AA batteries with normal use.

User Level Interface

At the user level, many things that happen under the hood are unknown to the user. For the user, using this smart lock should be simple, intuitive, and efficient. Currently, our smart lock can read, authenticate, and unlock a door in less than a second, which is comparable or even faster than traditional hardware locks with physical keys. For the user, fumbling for keys or inserting keys in the wrong orientation is a nuisance that is practically eliminated. The user can simply tap his or her Android phone to the smartlock and the door will unlock.

Entering Room

A user will touch their Android phone to the NFC sensor, which sends a message to the cloud that contains the user and room number. The server checks if the user has access, and the appropriate response is sent back to the lock, which will unlock or remain locked. The cloud server will also login the user for that room, and increment the room's population count. Currently, there is no implementation to account for multiple people entering the room within one unlock cycle.

Leaving Room

When a user leaves a lab or room they will log out of the lab by pressing his or her phone to the door lock. The same message will be sent to the server as if the user was login in. The server then logs the person out, and then decrement the room population. If a user leaves a room but doesn't log out then and they try to access the room later on. They will still be able to access the labs but the counter and login information will not be accurate.

2 Figures and Schematic (Sam)

2.1 Block Diagram

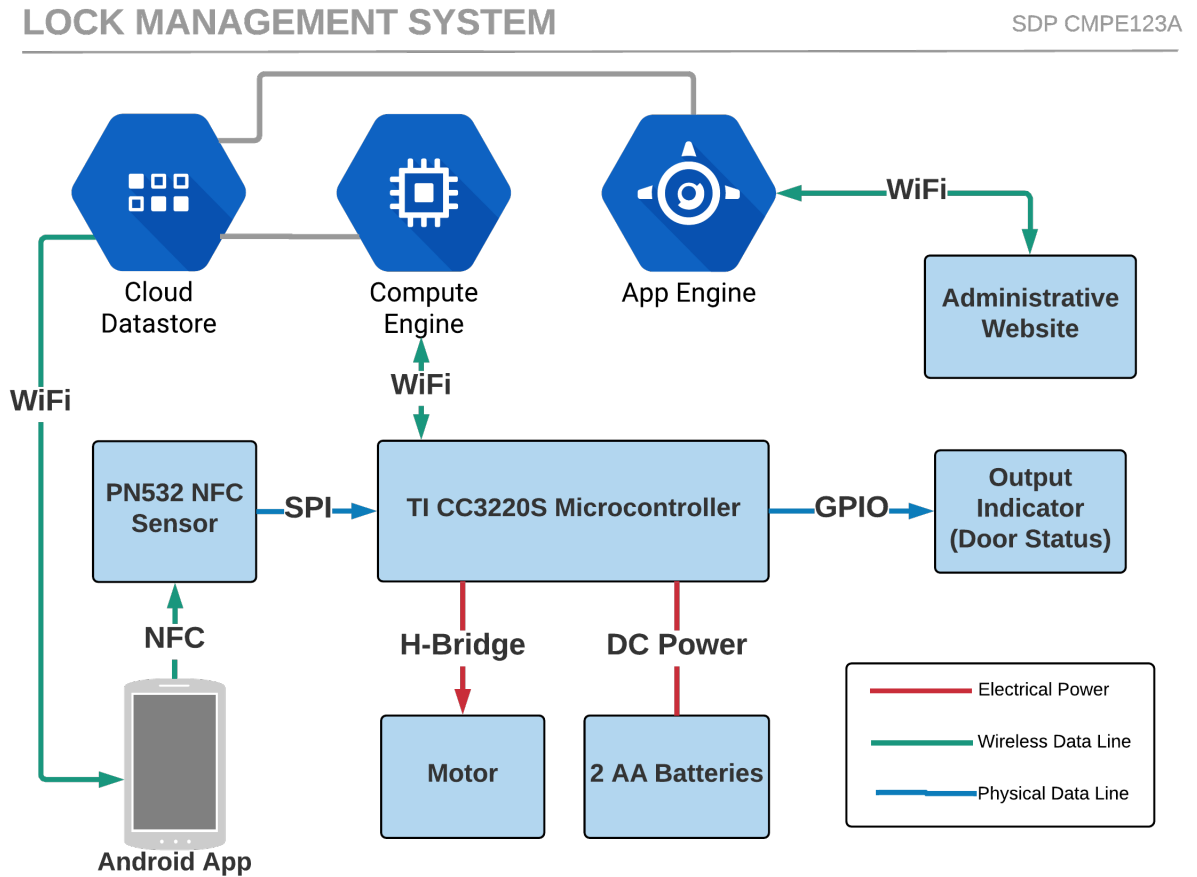


Figure 1: Block Diagram of System

2.2 Overall Software Flow Chart

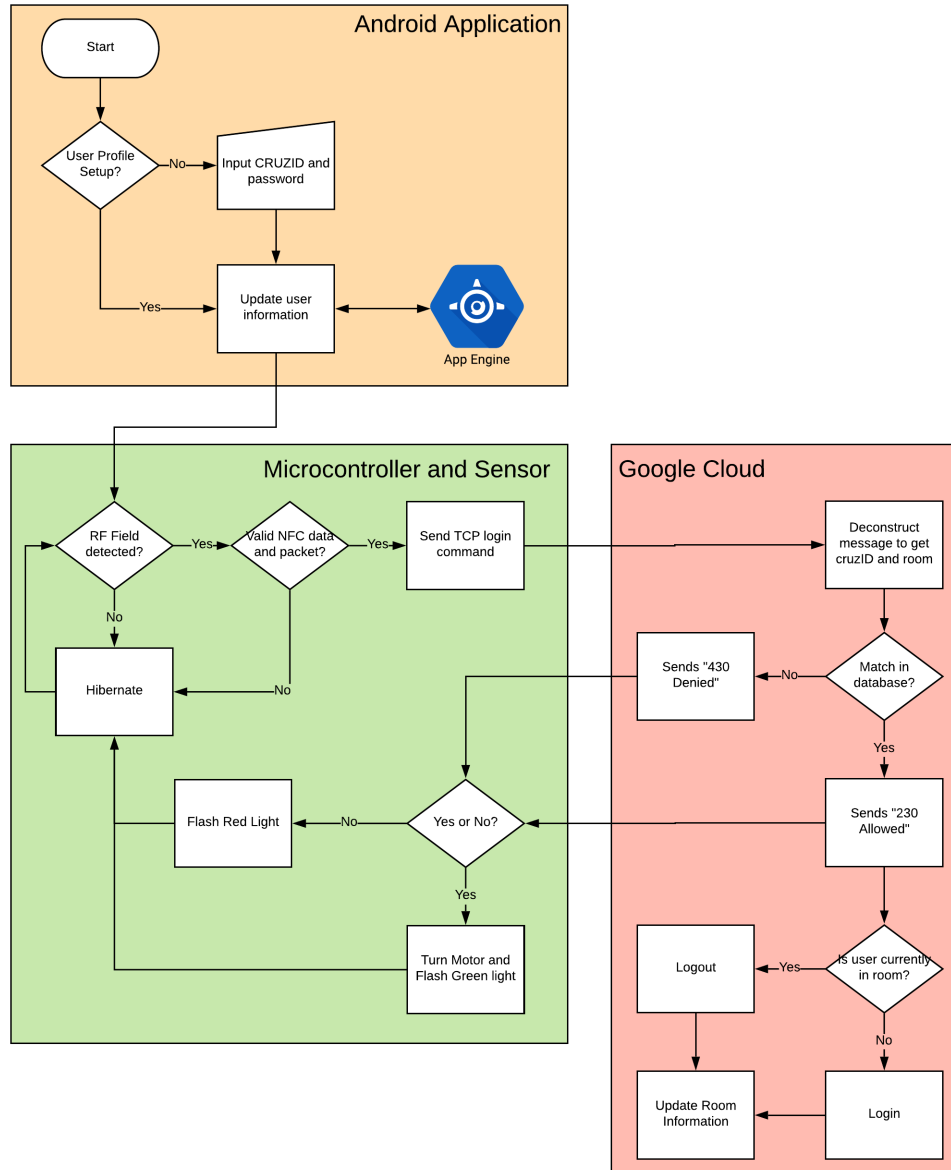


Figure 2: Yellow: Android app; Blue: NFC sensor; Green: microcontroller; Red: Google Cloud

2.3 Circuit Schematic

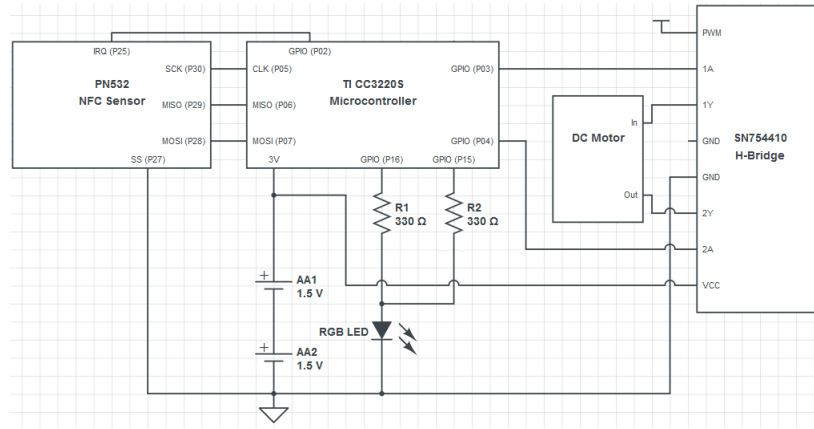


Figure 3: The circuit schematic of the project.

3 Microcontroller (Bowen)

3.1 Microcontroller Overview

The microcontroller is the center of the entire project, as it reads in NFC signals through the peripherals and acts as the network point for WiFi and further communication with the cloud. On the network side, after connecting to the Google Cloud Virtual Machine, the microcontroller connects to the Google Compute Engine, which returns the message code “230 Allowed” or “430 Denied”, depending on access. The microcontroller periodically wakes up from low power mode in order to maximize battery efficiency. A DC motor connected to the microcontroller will lock or unlock, in addition to flashing a green LED for unlock, and a red LED for denied access.

One important element of the microcontroller will be the cache. The cache is used so that the user can be let in faster. The cache will be small in size, with a size of approximately 16, and will be stored entirely locally. The main idea behind it is that users that frequently use the room will have their access information stored locally on the cache, and that way, the microcontroller would only have to search for the information locally instead of connecting to the cloud and waiting for a response. In a large system with thousands of users, the latency difference between a local cache search and a cloud search will be tangible.

3.2 TI 3220s API

3.2.1 StartWiFi()

```
void StartWiFi(void);
```

StartWifi() calls the Simplelink library to set up and start the WiFi. The WiFi connects to the pound defined network in “empty.h”. The network password must also be pound defined “empty.h”. This function created multiple threads with functions sl’task() and sl’sstart(), the two function that need to be called before any other function in the Simplelink library must be called. The function will try to connect at most five times before leaving the function.

3.2.2 IsAllowed()

```
int IsAllowed(char *cruzId, char *roomNumber);
```

Function sends the data using a TCP connection. The sent data is in the form of a string “\login cruzId roomNumber” to the gateway it waits for a return message. The to return types are “\logout” which means the user was just logging out or in the form of a message code “230 Allowed” which would unlock the door or “430 Denied” which flashes a red LED. If an error happens on the server side the micro will receive the message “\error”. When this happens the micro will attempt to send in the login message again.

3.2.3 YellowLight()

```
void YellowLight(uint8_t state);
```

YellowLight() turns on both red and green RGB LEDs, which give off a yellow color. This function is used exclusively for debugging. The argument passed in are either GPIOLED-OFF or GPIOLEDON, or 1 and 0 respectively. For example, this function is called after the microcontroller makes a successful WiFi connection but is trying to connect to the server. The microcontroller tries to connect to the server for five seconds before timing out, and during that time, the yellow light is on to indicate the server connection is taking longer than normal.

3.2.4 MotorControl()

```
void MotorControl(uint8_t direction);
```

MotorControl() controls the direction of the motor. The first argument defines the direction the motor turns, 0 for clockwise and 1 for counter clockwise. The motor will spin for one second before stopping.

4 NFC Sensor (Sam)

4.1 NFC Sensor Overview

The NFC Sensor we are using is the NXP PN532. It supports passive and active tag reading and writing, at ranges up to 4 centimeters.

4.2 Microcontroller and NFC Sensor interface

The command frame packet and acknowledgment from the sensor is handled by API calls as outlined in 4.3.

4.2.1 Command Frame Packet

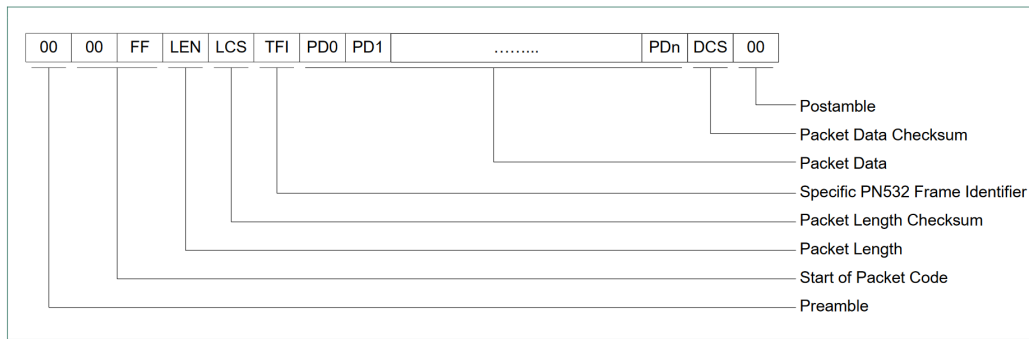


Figure 4: The command frame packet used to send and receive information over SPI

- Preamble: 0x00
- Command Code 1: 0x00
- Command Code 2: 0xFF
- LEN: Number of bytes in TFI, PD0, ..., PDn
- LCS: Checksum such that $LCS + LEN = 0x00$
- TFI: 0xD4 if MOSI, 0xD5 if MISO
- DATA:
 - PD0: command code
 - PD1: first data byte
 - ...
 - PDn: last data byte
- DCS: Checksum such that $TFI + PD0 + \dots + PDn + DCS = 0x00$
- Postamble: 0x00

4.2.2 SPI sequence for communication

1. Send a command
 - Write “Write Data”, 0x01
 - Write the command frame
2. Wait for ACK of PN532
 - Write “Read Status”, 0x02
 - Read one byte, PN532 is ready if byte is 0x01
 - Write “Read Data”, 0x03
 - Read 6 bytes of ACK frame (0x00 0x00 0xFF 0x00 0xFF 0x00)
3. Wait for Response of PN532
 - Write “Read Status”, 0x02
 - Read one byte, PN532 is ready if byte is 0x01
 - Write “Read Data”, 0x03
 - Read data bytes as outlined in command frame

4.3 Serial Peripheral Interface API Calls

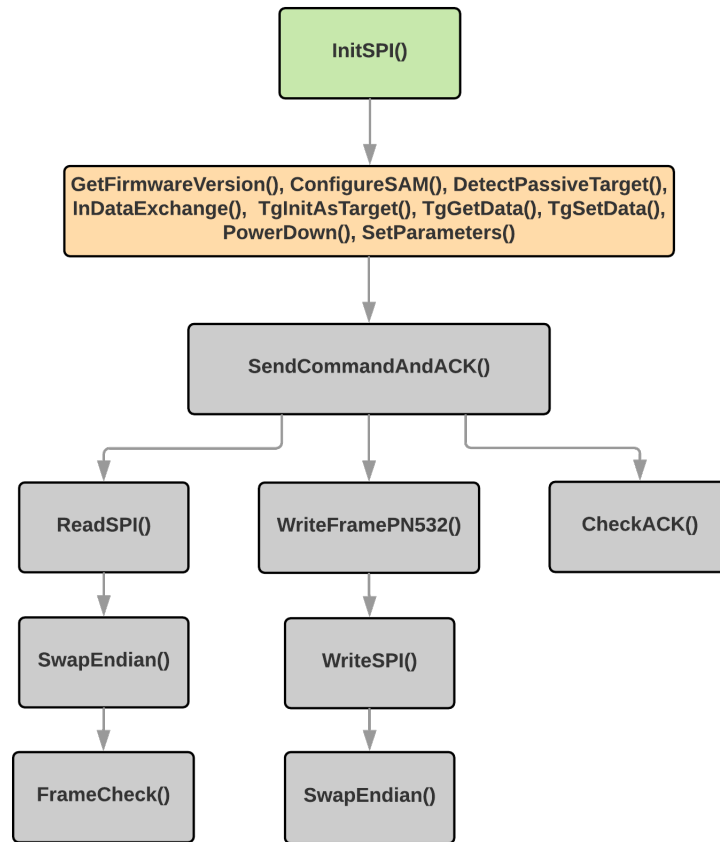


Figure 5: The functions the microcontroller calls to communicate with the PN532 sensor. Each functions has one or multiple subfunction under it.

4.3.1 GetFirmwareVersion()

```
uint32_t GetFirmwareVersion(void);
```

This function returns the firmware version of the PN532 sensor. This should be one of the first functions called just to initiate and make sure the SPI connection is working. Returns 0x0000 0000 if failure.

4.3.2 ConfigureSAM()

```
uint8_t ConfigureSAM(void);
```

Configures the Security Access Module (SAM). This function turns off the SAM, the default mode. Returns 0x00 if failure. This function needs to be called for proper sensor initialization.

4.3.3 DetectPassiveTarget()

```
uint32_t DetectPassiveTarget(uint8_t cardType);
```

Returns the card ID of the passive target. The ID is at most 4 bytes. Returns 0x00 if failure. Returns 0x02 if timeout.

4.3.4 InDataExchange()

```
uint8_t InDataExchange(void);
```

InDataExchange() is called after the sensor successfully detects a passive target. This function automatically makes a connection with the phone and exchanges authentication packets. It follows the ISO/DEP-14443-A protocol, which must be programmed into the phone and microcontroller beforehand. By default, the ID used on the phone is 0xF222222222, which is sent from the micro to the phone for verification. This value can be changed.

4.4 Sensor and Android phone interface

The actual communication between the sensor and the Android phone follows the ISO14443-3A standard. The Android phone acts as a passive card, and in figure 6, the block labeled “Mifare card” can be thought of as the Android phone.

Calling the function InListPassiveTarget() tells the sensor to create a RF field and look for passive targets within that field. The settings, such as baud rate or card type must be specified, and consequently, the sensor only looks for cards of that type. If the function is successful, it returns the 4-byte ID of the card. For an actual passive card, the ID is static, but for smartphones, the ID is a random 4-byte number.

After a successful passive pairing, the function InDataExchange() authenticates the phone and exchanges data. The sensor must send a predefined key that matches with that of Android phone. If the frame is correct and the keys match, the phone has 51ms (as defined in the PN532 user guide) to send its data before the NFC connection between the sensor and phone times out.

After reading from the phone, the sensor must write to the phone to terminate the connection. This is essential because closing the connection will mean a successful data exchange. The mobile application is programmed to vibrate the phone after a successful data exchange, which gives the user tangible feedback as to whether or not the sensor read his or her cruzID.

For our application, the sensor detects the passive phone and then tries to pair with it with the predefined key of 0xFFFFFFFF. From there the phone sends the corresponding cruzID of the user that is logged in. If the sensor successfully reads the cruzID, the phone will vibrate. An important thing to note is that a vibration does not mean that the user is allowed into the room. A vibration only means that the sensor correctly read the cruzID.

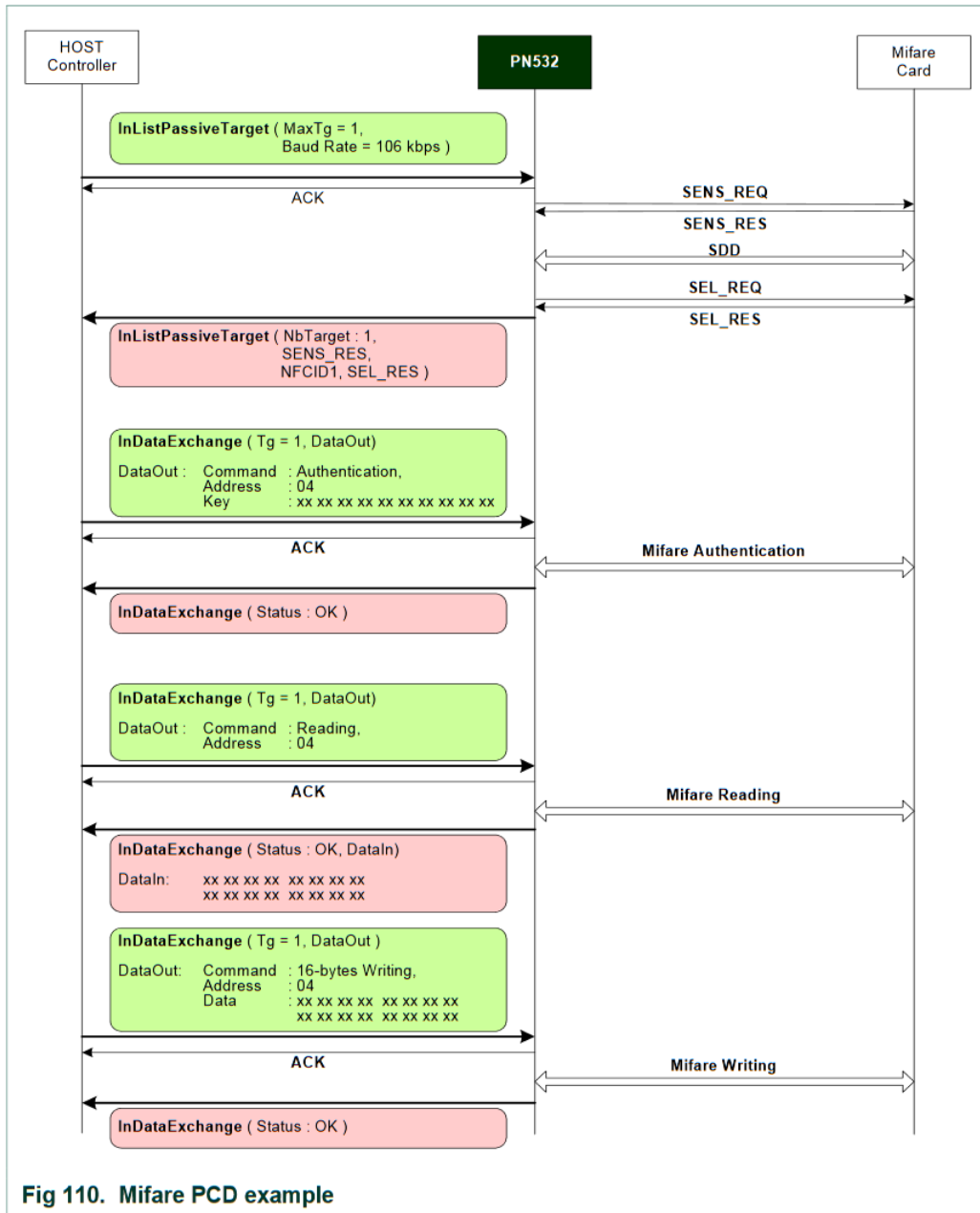


Figure 6: The communication sequence between the microcontroller and the Android phone, with the NFC sensor as the intermediary. On the right side of the figure is the Mifare card, which is a passive card that the Android phone emulates. The important elements are the authentication process, timeout, and closing the connection, all of which are described in this section.

5 Network Structure (Bowen)

5.1 Network Overview

The network portion of the system involves using Google cloud compute acting as a server for the micro. The website and android App will talk to the database directly. The overall network structure can be seen in figure ??.

5.2 Google Compute Engine

This a virtual machine running in the cloud. It has a static IP address and will act as the server for the door locks. The connection will be using TCP and SSL for security. The micro will send the room number and the cruzID from the android App. When a new connection is established the server start a thread to handle the login and logout from the micro. This allows the server to handle multiple request. The VM will run a query and send back either a “230 Allowed” or “430 Denied”. Once it sends the results it kills the connection and the thread. It then logs the results into the Datastore for the given Room.

5.3 Message layout protocols

Any message being sent to and from the cloud will follow the given format

”\[message type] [other data]”.

This way the cloud and micro can easily check for the message type and know how to handle it. A login message is in the form of **”\login cruzId room`number”**

5.4 Google Compute Engine API

5.4.1 get_student_classes()

```
def get_student_labs(cruz_id)
```

This function gets all of the classes thats the student is currently allowed access to.

5.4.2 get_classes_for_room()

```
def get_student_labs(room_number)
```

Function returns a query from the Google cloud which list all of the classes that currently have access to the given room.

5.4.3 login`room()

```
def login_room(room_number, cruzID):
```

This function is called when a user enters a room. The first argument is the room number of which the user enters and the second argument is the cruzID of the user that enters

the room. The function logs the enter time and stores it in the cloud. The leave time and cumulative time spent in the room is null.

5.4.4 logout_room()

```
def logout_room(room_number, cruzID):
```

This function is called when a user leaves a room. The first argument is the room number of which the user leaves and the second argument is the cruzID of the user that leaves the room. The function logs the leave time and stores it in the cloud. The cumulative time is calculated based on the leave time and the last time the student entered the room.

5.4.5 change_room_capacity()

```
def change_room_capacity(room_number, entering):
```

This function takes in the room number and the a boolean on the direction of the user. From their it will update the current capacity of the room by 1. This function doesn't handle the situation where a user lets multiple people in the room at one time.

5.4.6 reset_room_capacity()

```
def reset_room_capacity(room_number):
```

This function takes in the room number. It will reset the capacity of the room in the datastore back to zero.

5.4.7 reset_room_log()

```
def reset_room_log(room_number):
```

This function takes in the room number and deletes all the logs for that given room.

6 Google Cloud (Bowen)

Students, Faculty and rooms are all stored in Google Cloud. The cloud will respond from results from the microcontroller which will determine if a user has access to the rooms. The cloud storage can only be updated and from the administrative website. Students and rooms can be updated at anytime giving real-time access to the rooms. The cloud also logs all the login information for each room and which can be view on the administration website.

6.1 Google Cloud Entities (Sam)

6.1.1 Room Entity

The room entity lists every lock-equipped room. This entity will hold the room location, which specifies the building and room number, along with all groups and individual users that have access to it.

Room Entity			
ID	Room #	Groups	Users
name=indexE2-399	E2-399	["BELS","AMS147","CMPS101"]	jsmith
name=indexBE340A	BE340A	["CMPE123A","CMPE123B","CMPE129B"]	

6.1.2 User Entity

The user entity lists every user in the system. Additionally, each entry will include the cruzID of the user and the groups the user is in.

User Entity			
ID	Name	cruzID	Groups
name=indexSAZWU	Samuel Wu	sazwu	["AMS147","CMPS12B"]
name=indexBOJBROOK	Bowen Brooks	bojbrook	["CMPE123A"]

6.1.3 Room Logs Entity

The room logs entity will log entry and exit times for each user. There will be a cumulative time for each unique user in the room. The log in each room can be cleared and the enter time, exit time, and cumulative time will be reset. Each room will have its own log.

BE340 Log Entity					
ID	cruzID	Enter Time	Exit Time	Delta time	Cumulative
name=sazwu0	sazwu	02-07 17:26:55	02-07 17:27:02	0:00:07	0:00:07
name=bojbrook0	bojbrook	02-07 17:26:55	02-08 11:30:22	18:03:26	18:03:26
name=sazwu1	sazwu	02-08 11:27:19	02-08 11:27:57	0:00:44	0:00:51
name=hello0	hello	02-08 11:31:02	null	null	null

7 Administrative Website (Sam)

7.1 Administrative Website Overview

The administrative website allows the administrators to add, remove, or modify access for users and rooms. The website offers a live population count of each room and other useful data, such as histograms of enter and exit times, or average time spent in a room. The Website is using the Google App Engine as its back-end and Flask as the front-end. This allows for the website to make direct calls to the datastore without the need of a server.

7.2 Administration API

7.2.1 AddNewUser()

```
def add_new_user(index, name, cruzID, priority, class_list = []):
```

This function adds to the “User” entity. The first argument is the index in the database, which acts as the key in case it needs to be accessed later. The second argument is the name of the student, with first and last name separated by a space. The third argument is the cruzID of the student without the `ucsc.edu` following it, i.e. `sazwu`. The fourth argument is the user priority. Highest priority is given to the lowest number. For now, admin priority = 0, faculty priority = 1, grad priority = 2, and undergrad priority = 3. The last argument is a Python list of classes the student is in i.e. `CMPE167`, `AMS147`.

7.2.2 AddNewRoom()

```
def add_new_room(index, room_number, class_list = []):
```

This function adds to the “Room” entity. The first argument is the index in the database, which acts as the key in case it needs to be accessed later. The second argument is the room number of a room that has an a lock on it, i.e. `BE301`, `BE340A`, `E2-599`. The third argument is a Python list of classes that have access to the room i.e. `CMPE123A`, `CMPE121`, `CMPE167L`.

7.2.3 ModifyRoomEntity()

```
def modify_room_entity(lab_index, room_number, class_list = []):
```

This function modifies an existing “Room” entity. The entries in the entity must all be specified in the arguments. The first argument is the index in the database. The second argument is the room number in a room with an accessible lock. The third argument is a Python list of classes that have access to the room.

7.2.4 ModifyUserEntity()

```
def modify_user_entity(user_index, name, cruzID, class_list = []):
```

This function modifies an existing “User” entity. The entries in the entity must all be specified in the arguments. The first argument is the index in the database. The second argument is the first and last name of the student separated by a space. The third argument is just the cruzID of the student, and the fourth argument is a Python list of classes the student is in.

7.2.5 QueryAllRoomsEntities()

```
def query_all_rooms_entities():
```

This function simply prints all “Room” entities and the information in each.

7.2.6 QueryAllUserEntities()

```
def query_all_user_entities():
```

This function simply prints all “User” entities and the information in each.

7.2.7 DeleteRoomEntity()

```
def delete_room_entity(lab_index):
```

This function deletes an entire “Room” entity specified by the index passed in as the first argument.

7.2.8 DeleteUserEntity()

```
def delete_user_entity(lab_index):
```

This function deletes an entire “User” entity specified by the index passed in as the first argument.

7.2.9 LogInRoom()

```
def login_room(room_number, cruzID):
```

This function is called when a user enters a room. The first argument is the room number of which the user enters and the second argument is the cruzID of the user that enters the room. The function logs the enter time and stores it in the cloud. The leave time and cumulative time spent in the room is null.

7.2.10 LogoutRoom()

```
def logout_room(room_number, cruzID):
```

This function is called when a user leaves a room. The first argument is the room number of which the user leaves and the second argument is the cruzID of the user that leaves the

room. The function logs the leave time and stores it in the cloud. The cumulative time is calculated based on the leave time and the last time the student entered the room.

7.2.11 CountRoomPopulation()

```
def count_room_population(room_number):
```

This function counts the number of people in the room specified by the first argument. It counts the number of people by counting the number of indexes in the room with a null value in the exit time.

7.2.12 ClearRoomLog()

```
def clear_room_log(room_number):
```

This function clears the entity and log of the room number specified in the first argument. This function could probably be used at the end of the day or end of the quarter.

7.2.13 HistogramEnterTime():

```
def histogram_ente_time(room_number):
```

This function gives a Python list of size 24 with the frequency of time entered. Each hour interval corresponds to an index in the list. The function requires the first argument to be the room number of the room of which the histogram is desired.

7.2.14 HistogramExitTime():

```
def histogram_exit_time(room_number):
```

This function gives a Python list of size 24 with the frequency of time exited. Each hour interval corresponds to an index in the list. The function requires the first argument to be the room number of the room of which the histogram is desired. Any user in the room will have a “null” value for exit time, and thus will not be counted.

7.2.15 AddClassToRoom()

```
def add_class_to_room(room_number, classes = [])
```

This is a helper function for parsing the CSV file when using function ParseCSVRoomEntity(). It takes in the room number and the classes to add. For example, if room “BE-340” can now be accessed by classes “CMPE167 and AMS114”, the function would

take in “BE-340” as the first argument and the list “CMPE167 and AMS114” as the second argument.

7.2.16 ParseCSVRoomEntity()

```
def parse_csv_room_entity(file)
```

This function parses a CSV file to create and add to a room entity. The first row of the .csv file is not read, as it is meant to label the columns. The first column of the file should be the room number and the second column of the file should be a list of classes that can access the room, separated by spaces.

7.2.17 CSVToCloudRoom()

```
def csv_to_cloud_room(file)
```

This function calls ParseCSVRoomEntity() and adds the data from the .csv file to the “Room” entity. The first row of the .csv file is not read, as it is meant to label the columns. The first column of the file should be the room number and the second column of the file should be a list of classes that can access the room, separated by spaces. The only argument should be the .csv file name.

7.2.18 AddClassToUser()

```
def add_class_to_user(user, info_list = [])
```

This function adds a list of classes to the user, and is used as a helper function for ParseCSVUserEntity() and CSVToCloudUser().

7.2.19 ParseCSVUserEntity()

```
def parse_csv_user_entity(file)
```

This is a helper function used by CSVToCloudUser(). Refer to section CSVToCloudUser().

7.2.20 CSVToCloudUser()

```
def csv_to_cloud_user(file)
```

This function takes in the data from a .csv file and puts it directly in the “User” entity. The .csv file should have the information name, cruzID, priority, and classes in that order. If there are multiple classes, the classes can be separated by spaces while staying in that column.

8 Android Application (Bowen)

8.1 Android Application Overview

The android application allows for a user to sign in using their CruzID and password. When the phone is tapped against an NFC sensor the application will transmit the CruzID to the microcontroller. The application will have access to the Google cloud in order to view which rooms they have access to. To access the datastore the Android App will use HTTP calls to the administration website to get the list of labs available to them. The pages for the Android APP are not accessible to the administration and can only be accessed by the Application.

<https://lock-management-system.appspot.com/android> is the base call for all the HTTP commands from the Android App.

8.2 Android Application API Calls

8.2.1 GetListOfLabs()

```
ArrayList<String> GetListOfLabs(String cruzId):
```

This function asks the Google datastore using HTTP calls to get the list of labs the user has access to with the cruzID. It return a list of all the labs they can get into for the quarter.

8.2.2 GetLabsCapacity()

```
ArrayList<Integer> GetLabsCapacity(String cruzId, ArrayList<String> Labs):
```

This function asks the Google datastore using HTTP calls to get the current capacity for each lab. The cruzId will be passed in to ensure that the user has access to the lab. The function return an Arraylist that contains the current lab capacity.

8.2.3 GetRoomLogs()

```
void WriteCruzId(String cruzId, String Lab):
```

A function that generates an http call to the website which responds with a json file of all the logs for that specific user. Which is then displayed to the user.

9 Battery (Sam)

9.1 Battery Overview

The battery consumption is theoretical at this point, and is based off of a weighted sum of the manufacturer's specification in the datasheet for the TI CC3220S microcontroller and the NXP MFRC522 sensor. $amps_{total}$ is the weighted sum for amps used in a day.

9.2 Battery Calculations

$$\begin{aligned} & amps_{total}(\#logs) \\ &= \frac{(\#logs)(time_{log})}{60 * 60 * 24s} (amps_{log}) + \frac{60 * 60 * 24s - (\#logs)(time_{log})}{60 * 60 * 24s} (amps_{sleep}) \end{aligned} \quad (1)$$

$time_{log}$ is the total time it takes to do one complete log, including the time to turn sensor on, transmit over WiFi, and then receive over WiFi.

$$time_{log} = time_{sensor} + time_{tx} + time_{rx}$$

$$\begin{aligned} & amps_{total}(\#logs) \\ &= \frac{(\#logs)[(WiFi_{tx})(time_{tx}) + (WiFi_{rx})(time_{rx}) + (sensor_{on})(time_{sensor})]}{60 * 60 * 24s} \\ &+ \frac{60 * 60 * 24s - (\#logs)(time_{log})}{60 * 60 * 24s} (sensor_{sleep} + micro_{sleep}) \end{aligned} \quad (2)$$

We conducted multiple measurements, and found that the transmit and receive time were within specifications of the datasheet. However, we took an overestimate of the times it takes to transmit and receive, with transmit = 0.5s and receive = 2s. Additionally, we overestimated the time it takes for the sensor to read, at 0.5s. According to the datasheet, the data transfer over NFC must occur within 51ms, or else a timeout occurs.

Either way, the “active” power consumption of the device, when it reads in over NFC and exchanges data over WiFi, is negligible when compared with the idle power consumption.

$$\begin{aligned} amps_{total}(\#logs) &= \frac{(\#logs)[(266mA)(0.5s) + (53mA)(2s) + (100mA)(0.5s)]}{60 * 60 * 24s} \\ &+ \frac{60 * 60 * 24s - (\#logs)(5s)}{60 * 60 * 24s} (3mA) \end{aligned} \quad (3)$$

$$amps_{total}(\#logs) \approx 3 * 10^{-3} + 3.47 * 10^{-8}(\#log) \quad (4)$$

AA batteries are typically anywhere from 2000 – 3000mAh, so taking the lower bound, our 2 AA batteries in series will have approximately 4000mAh worth of charge.

$$time_{months}(\#logs) = \frac{1}{24 * 30} * \frac{amphour}{amps_{total}} \quad (5)$$

$$time_{month}(\#logs) = \frac{5.555 * 10^{-3}}{3 * 10^{-3} + 3.47 * 10^{-8}(\#logs)} \approx \frac{160102}{\#logs + 86455}$$

9.3 Battery Graph

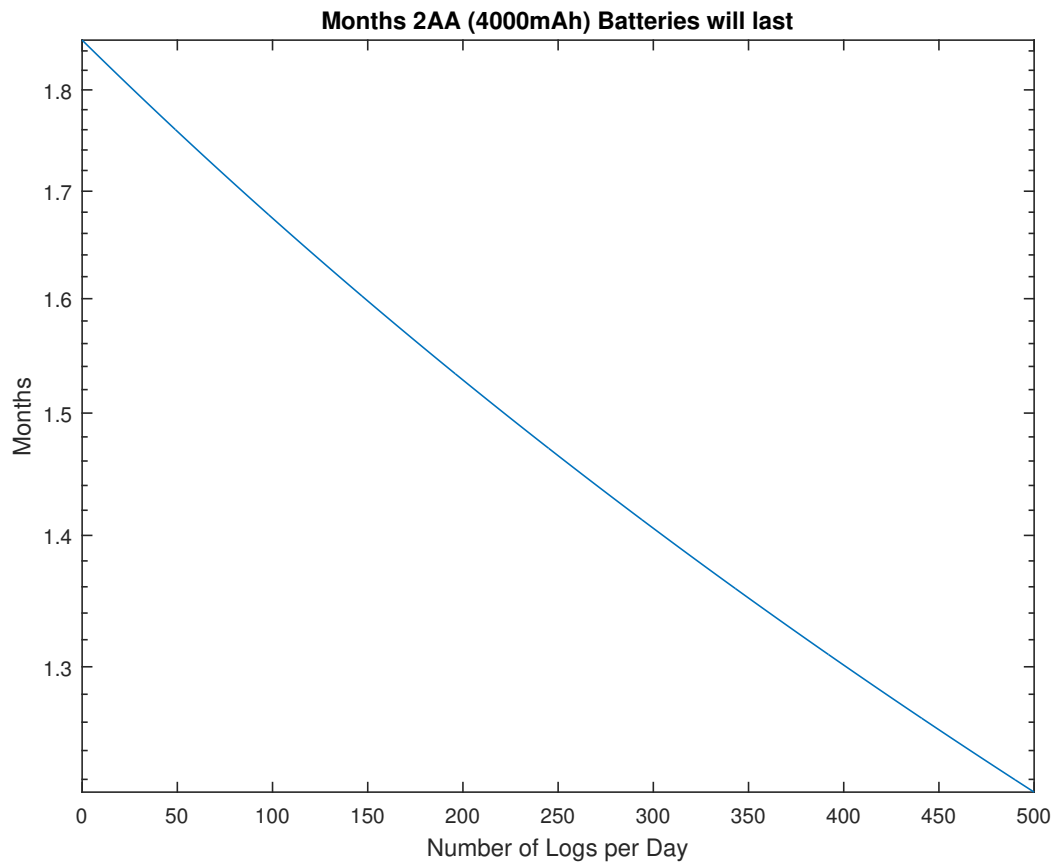


Figure 7: Theoretical Battery life using 2 AA batteries (4000mAh) and the NXP MFRC522 Sensor

10 Schedule (Bowen & Sam)

10.1 Winter Quarter

10.1.1 Quarter Goals

By the end of the quarter, the cloud and microcontroller should be able to transmit and receive information from each other. Additionally, the microcontroller should be able to differentiate unique RFID tags.

10.1.2 Tasks for the Quarter

1. RFID/MCU communication
2. Cloud/MCU communication
3. Database
4. LEDs

Bowen	
Week 4	Design database, Design cloud API, Populate database
Week 5	Design cloud API
Week 6	Cloud AUTH/access, Design cloud API
Week 7	MCU Push/pull database, MCU Log interaction
Week 8	Cloud push results to MCU
Week 9	Finish MCU, Cloud clean up
Week 10	Start API calls for website

Sam	
Week 4	Learn basic Google Cloud, Design database, Design cloud API
Week 5	Populate the database
Week 6	MCU internet access, SPI interface for sensor
Week 7	MCU/RFID communication (PN532)
Week 8	Write PN532 Library (read/write/wakeup/sleep)
Week 9	Debug PN532 Library
Week 10	Get RFID tag data, RFID type

10.2 Spring Quarter

10.2.1 Quarter Goals

Everything should be completely finished. This includes full functionality of the Android App for communicating with the cloud and the NFC sensor with a *NFC* signal. Additionally, there will be an administrative website that pulls data from the cloud. The website will have administrative functionalities such as: adding, removing, or modifying user privileges. The administrator will also be able to navigate a clean UI to view analytics and data.

10.2.2 Tasks for the Quarter

1. Power Management
2. DC Motor/H-Bridge
3. Website
4. NFC/App communication
5. Web/Cloud communication
6. App/Cloud communication

Bowen	
Week 1	Familiarize with Android API, Design UI
Week 2	NFC/App communication
Week 3	NFC/App comm, App/Cloud comm
Week 4	App/Cloud comm, start design for website
Week 5	Finish app, cleanup app UI, website UI
Week 6	Website/cloud comm, website UI
Week 7	Finish website, debug all comm
Week 8	Debug all comm, debug any small things
Week 9	Buffer week, start report/presentation
Week 10	Finish everything

Sam	
Week 1	DC Motor/H-bridge
Week 2	App/NFC comm, clean up MCU code
Week 3	App/NFC comm, App/Cloud comm
Week 4	App/Cloud comm, power
Week 5	Power Consumption
Week 6	Website UI, Web/Cloud comm
Week 7	Cleanup App UI, website UI
Week 8	Debug everything, buffer week
Week 9	Start report/presentation
Week 10	Finish everything

11 List of hardware components (Bowen)

Component	Cost	Quantity	Total
TI CC3200S	\$39.99	3	\$119.97
Battery Case	\$1.50	2	\$3.00
Op Amp	\$0.95	4	\$3.80
Resistor Kit	\$7.95	1	\$7.95
NFC Sensor MRFC522	\$9.99	0	\$0.00
AA battery 20 pack	\$8.54	1	\$8.54
NFC Sensor PN532	\$12.99	2	\$25.98
DC Motor	\$1.95	2	\$3.90
LEDs 5 Pack	\$2.95	1	\$2.95
H-Bridge	\$2.35	2	\$4.70
Total Cost			\$180.80

11.1 Video Link

[Video Demonstration](#)