iDoorlock

Author: Alex Li, Alex Xu, Michael Chen: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—An NFC-based smart door lock that allows users to access their home with only their phones. iDoorlock features two-factor authentication using established smartphone technology with 100% accuracy to prevent malicious entry and a rapid deadbolt unlocking mechanism that unlocks the door in under 5 seconds to allow seamless entry and exit. All user data is managed and processed within AWS for cost efficiency, built-in security policies, and ease of use.**

Index Terms—API Gateway, AWS, DynamoDB, Lambda, NFC (alphabetical order)

I. Introduction

Though technology is advancing at an unprecedented pace, there are still many elements of modern life that remain archaic such as the burden of carrying keys for your home. Keys are small, easy to misplace, and difficult to find when lost. Our project, iDoorlock, aims to streamline the door opening process and remove the need for keys while still maintaining a high level of security and quick access times. iDoorlock uses NFC, or near-field communication, technology in phones to be able to unlock locked doors, similar to an RFID or Bluetooth mechanism. Phones are an integral part of modern life, and they are much easier to find if misplaced through GPS tracking functionality. With just a hover of a user's phone, the NFC reader and the lock terminal will interact, and the terminal will correctly authenticate the user or maintain the lock if the user doesn't have the right credentials. iDoorlock is inspired by Apple Pay, a popular payment system with which customers can just tap their phone to a payment terminal for each monetary transaction. We wanted to do something similar to the lock system for doors, effectively making unlocking doors a lot faster and seamless. Besides speed and convenience, we also want our prototype to be secure and marketable, so people would actually be willing to use it. The most important benefit of using a technology like NFC as opposed to just having a standard smart lock is to negate most remote unlocking mechanisms. In 2016, a University of Michigan team demonstrated how they could remotely snoop on the traffic between a smart lock and a connected Samsung SmartThings phone application, allowing a hacker access to the code and the ability to remotely change the PIN code without proper authorization. In contrast, NFC technology is a simpler way to unlock a door, and it also uses technology that already exists in smartphones, making it cost-effective for most Android and Apple smartphone users. iDoorlock specifically employs NFC technology's 4cm communication distance to ensure that a user has physical access to their phone, which makes wireless spoofing significantly more difficult.

We use a NFC reader for our lock terminal, which means we would only need to process the information from the NFC reader through a Raspberry Pi. We will be building the lock system including motorizing a deadbolt lock and connecting the Raspberry Pi to a web server (for authentication purposes). The phone and the NFC reader would interact, and information is authenticated in the Raspberry Pi, triggering a motor to unlock or do nothing.

II. Design Requirements

Given that the nature of our project is to secure a door, we must maintain the same level of security that a normal door lock provides. As such, iDoorlock requires that the door will not open 100% of the time for unauthorized requests and that the door will open 100% of the time for authorized requests. We will test this by touching an unauthorized phone to the NFC reader to ensure that the lock never opens. Conversely, we will also touch an authorized phone to the reader and ensure that the lock opens every time. On the AWS side, we are authenticating requests with IAM roles that have permission to invoke our REST API and Lambda. These IAM roles will not have any additional permissions, nor will AWS have admin level access. This is to ensure that malicious attackers are not able to compromise the integrity of the entire AWS infrastructure. To test the security of our AWS infrastructure, we will make requests with valid and invalid IAM access keys. We will ensure that any requests made with the valid IAM access keys will succeed and send the expected server response. For the requests made with the invalid IAM access keys, we will ensure that the server does not modify any user data and sends the expected failure response.

In addition to security requirements, we also need to ensure that the delay between a user touching their phone to the NFC reader and the door unlocking on a successful request is no longer than 5 seconds, as that is the average amount of time that it takes a person to open a keyed door lock. The latency that we measure includes the entire request/response lifecycle, from the request being sent to AWS by the Raspberry Pi to the lock being fully opened after receiving a success signal from AWS. We will test this by performing a number of trials in which we touch our authorized phone to the NFC reader and using a stopwatch to time the number of seconds it takes for us to be able to open the door.

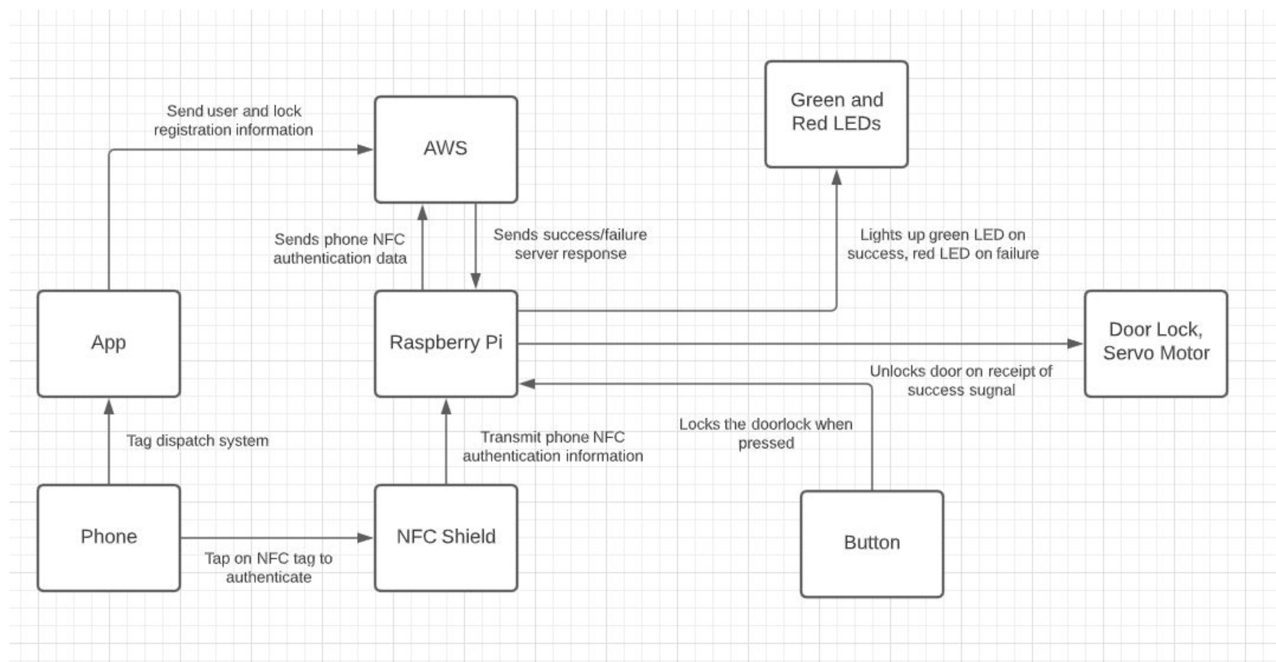


Fig. 1. iDoorlock system architecture diagram.

III. Architecture and/or Principle of Operation

Our system is divided into three distinct parts to allow for greater parallelization of work, which maximizes the team's efficiency in developing iDoorlock.

The lock hardware and circuitry is the first component within the system architecture. The outward-facing NFC Shield module waits for an NFC tag to forward to the Raspberry Pi. The Raspberry Pi is then responsible for sending the information to AWS and receiving a response. This response is then sent back to the Raspberry Pi, which acts on it to determine whether the servo motor connected to the lock should be powered or not. This physical hardware and circuitry system relies on routing information from different endpoints within the system quickly to produce a response for the actual door lock.

AWS is the second component within the system architecture. In the context of iDoorlock, AWS refers to three AWS services needed for our wireless authentication: Lambda, DynamoDB, and API Gateway. Any wireless requests from our lock hardware and circuitry system are received by AWS and then acted upon. Regardless of the type of request, AWS API Gateway will forward the request information along to Lambda and DynamoDB, but the latter two components interact differently depending on if the request pertains to the activation of a new lock product or to a general unlocking request. After the request is processed, the AWS system will then send the response back to the lock hardware and circuitry system or the phone application system. API Gateway requires REST API endpoints to use https protocol, which adds security against man-in-the-middle attacks and eavesdropping.

The phone application is the last subsystem within the system architecture. This application acts as the main interface through which a user will interact with iDoorlock's other two subsystems. The detection of the NFC tag on the NFC Shield Module will cause the Android Tag dispatch system to start a service to handle messages sent via NFC. The phone application also allows for the registration of a new iDoorlock. Information about users and locks is displayed on the app and will be available via communication with the server. This is the main user-level abstraction through which a majority of the important information will be displayed, including a list of verified locks and a detailed lock attempt history.

Since the Design Review, we have made some significant changes to our design. We removed the Arduino Uno as we discovered that the Raspberry Pi would be able to drive the servo motor with output PWM signals, thus removing the need for the Arduino; we will discuss this later in our design trade studies. We also added a button and green and red LEDs. The LEDs indicate to the user whether or not the NFC reading was successful or not, and the button can be pressed to lock the lock. Alternatively, the lock will automatically lock itself after five seconds. Lastly, we changed from using WebSockets to a REST API model, as we encountered issues integrating WebSockets within our door unlocking Python script with our C NFC service.

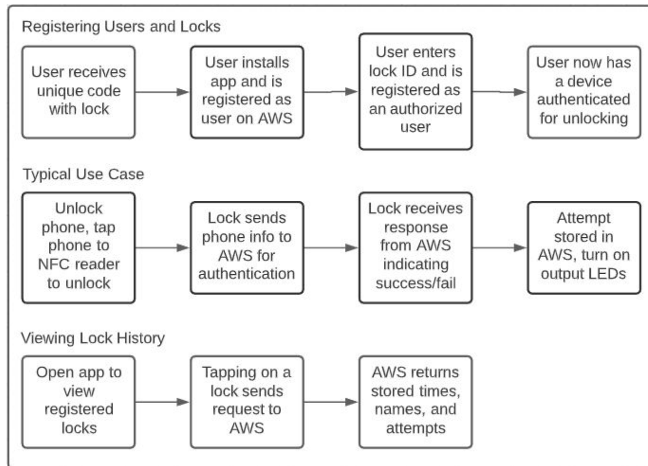


Fig. 2. Lock registration, typical use, and lock history flow diagrams.

IV. Design Trade Studies

A. *Authentication Methodology*

To verify the identity of a user, we had originally wanted to go with a depth sensing camera that could do 2D and 3D imaging, the Intel RealSense Depth Camera SR305. Powered by USB Type B, this small $80 camera would have been the main point of interaction with the user in our design, and it would be at the physical front of our iDoorlock system. With the combination of OpenCV running on our Raspberry Pi and the data output from this camera, we set a 3% false positive and 7% false negative rate for facial recognition. During the design process, we had these metrics as we felt reasonably confident in the accuracy of our system to be able to identify users attempting to gain access and, through comparison of both 2D and 3D imaging of a user, be able to very accurately verify identity. However, it was pointed out during our project proposal that a lock that is not 100% safe fails to do the job of a lock better than a regular door lock and key. Aside from security concerns, other issues were raised about this design, including user liveness, angle of imaging, and general accessibility, as a fixed camera may not be able to see every person of every height and age, and simply attempting to match an image and a depth map can be fooled.

Instead, after receiving this criticism, we pivoted to use the HiLetGo PN532 NFC Module Kit, which cost just under $10. The goal of our design pivot was to replace our identification method with something more versatile but also more secure, and we realized that, with smartphones being nearly ubiquitous in society, taking advantage of technology already existing in the pockets of most people would be a smarter solution. This swap achieves a better value by not only being more compact, it is also cheaper and can allow us to prompt a user for state-of-the-art biometric identification techniques already present in most smartphones, like a fingerprint scanner. All that we would need to do is to modify the phone application we were planning to use to display lock histories and ask the NFC module to trigger the application to open. The opening of the application will coincide with a trigger for fingerprint authentication to log into the application, which will serve as a basis for authentication. By requiring a device to be physically present next to the NFC module and a valid fingerprint, our solution is significantly smaller, cheaper, smarter, and more secure.

B. *AWS*

For the AWS component of our project, we chose to use API Gateway to set up the REST API endpoints, Lambda for serverless computing, and DynamoDB for user data management. There were a number of alternatives that were considered, such as Microsoft Azure and GCP for the cloud provider. We decided to use AWS over the other options because of the AWS credits provided to us by the university. In addition, the team has prior experience with AWS, which will allow us to accelerate our project schedule and have additional slack in the event of unforeseen delays.

Within AWS itself, we considered EC2 to host a server that constantly monitors API requests instead of a serverless model with Lambda. We chose to use Lambda over EC2 because the nature of our project means that we will not be handling a large number of requests on a daily basis. The average person does not open their front door more than 5 times a day, and having an EC2 instance running an entire day to handle 5 requests per lock is extremely inefficient. With the serverless model, we pay only for each request that we make to the Lambda rather than the instance runtime of a server.

For our database solution, we considered other database options such as hosting a MySQL or MongoDB server on AWS. We decided on DynamoDB because of the ease of integration with AWS Lambda and cost concerns. The pynamodb Python library allows us to easily connect our Lambda function to our DynamoDB table and manipulate our user data. In addition, setting up a DynamoDB table is extremely simple. New tables can be created in the AWS developer console in a matter of minutes. The estimated cost of running an RDS (Relational Database Service) cluster on AWS is $850 a month. On the other hand, AWS provides 25 GB for free as part of their Free Tier and there are no additional read/write rate limits or quotas. In addition, setting up a RDS cluster requires secure password management, an issue that DynamoDB does not have.

Exclusively using AWS services to handle our data management and request handling also aids in accelerating development. The AWS SDK for Python allows us to manipulate multiple services with minimal code, and the existing documentation that Amazon provides is extremely comprehensive and easy to parse.

C. *Android Phone*

NFC technology is now common on most modern smartphones. Due to time constraints, we decided to narrow down our design to work against the most common phone type. With this constraint, we would need to choose among the two most popular types of phones, iPhone and Android. We ended up choosing Android for a number of reasons. First off, Android currently has the majority of the market

share, with the iPhone coming in a close second. We wanted our design to work with as many phones as possible to prove our design could work with the common phone. Another important reason why we chose to use an Android Phone in our design is because the Android has well documented and open source NFC frameworks and technology. On the other hand, the iPhone's NFC technology is not developer friendly, and they hide their Apple Pay technology to prevent competition. To further elaborate, in order to develop our application, we would need peer to peer NFC, which is only available to the Apple Pay software on iPhone. So developing our application on iPhone would need more work and effort to emulate peer to peer NFC in a seamless way.

D. *Removing the Arduino Uno*

In our original design review report, we reference the use of an Arduino Uno to act as an communication interface between the Raspberry Pi, our NFC module, and the servo motor and lock hardware piece. Our original line of thinking was that our familiarity with the Arduino Uno would make it easier to interact with the NFC module, as that was the original microcontroller intended for the module, and use the pinout on the Arduino to connect to both it and the servo motor mechanism. Our system would then use the USB cable included with Arduino Uno to connect to the Raspberry Pi. This way, incoming power into the Raspberry Pi would also power the Arduino Uno. As the two were connected via USB, it would also be relatively easy to communicate between the two microcontrollers.

However, through the design process, we quickly realized that the Arduino Uno would become rather useless. As the Uno's original intention was to be a glorified message passer between the three subsystems in our overall design, we did some research and experimentation to see if connections between the Raspberry Pi and the rest of the hardware was doable. We quickly found out that the SDA/SCL pins on the Raspberry Pi would work perfectly for the I2C communication between the NFC module and our Raspberry Pi, and that the plethora of 5V, GND, and GPIO pins work fine for any circuitry-related implementations that we would later pursue (specifically, the LEDs and button, which work fine on any pin, and the PWM output signal to drive our servo motor, for which the Raspberry Pi has a specific "CLK" GPIO pin).

The removal of the Arduino Uno is an important revision in our design. Functionally, nothing changes from our original vision of the product. However, any communication delays that would have resulted from the Arduino Uno waiting for messages among various systems has been systematically eliminated, and the Arduino Uno itself being removed reduces the number of attack vectors which a hacker could look to exploit. Finally, it also reduces the size of the overall system, which played a big role in getting our lock housing to be in as small of a form factor as possible for the best possible end product.

E. *Unlock vs Biometrics*

In our original design review report, we planned to use biometrics as a method of authenticating that the correct user had the phone. We changed our design so that any unlocked state would be valid for authentication. We did this for several reasons. One of the primary reasons is that older models don't have the technology to support biometrics. Because we aim to reach a wide audience, we decided that an unlocked state would be more practical. Another key reason is that we wanted to give the choice of biometrics or unlock to the user, so the user's choice of biometrics or pin number or always unlocked is representative of their choice of security. Another reason is that the Android development environment supports biometrics in apps, but restrictions are placed so that extra steps would be needed to support a pop up from a service (which is where the NFC communication is spawned from). This service runs in the background, and would need to run an Activity, which is a foreground interface, which requires unconventional solutions to achieve. Doing so would also remove some of the convenience factor, so with the other reasons, we decided to go with a simple unlock as opposed to biometrics for authentication.

V. System Description

A. *Lock System, Hardware, and Overall Design*

iDoorlock will use a TiankongRC MG995 servo motor in conjunction with an AmazonBasics Deadbolt lock as the physical lock hardware. From the lock, we will only need the actual deadbolt alongside the mounting plate, the mounting screws, and the inside cylinder. We will attach the servo motor to this inside cylinder at the lock tailpiece so that it can turn the cylinder and bolt together. This will be a 3D printed part so we can get the dimensions exactly as we want.

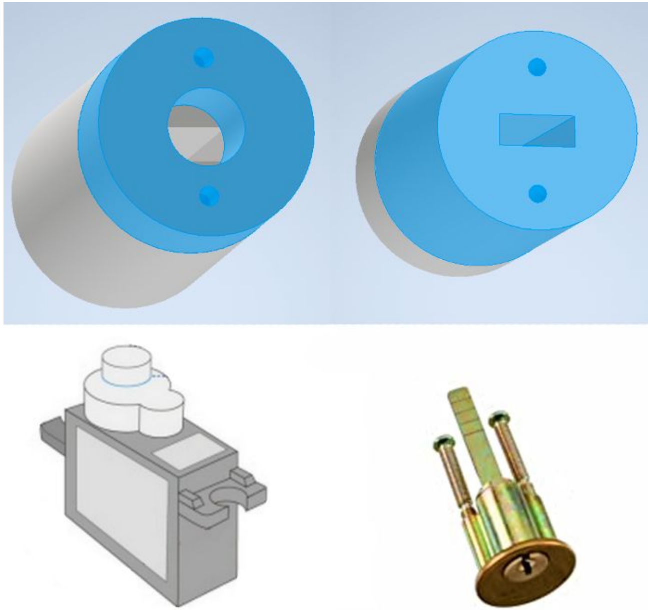


Fig. 3. Top: CAD design of the 3D printed servo lock-cylinder tailpiece connector. Bottom: the ends at which this part will be connected.

The hardware package of iDoorlock has this lock and servo motor combo at the end of its hardware system. Directly connected to the lock and servo motor combo is a Raspberry Pi. The Raspberry Pi is mainly responsible for all of the web-based connections to AWS, which will be discussed in the AWS section, but it also connects to the NFC Shield module and the lock hardware package via its pinouts, and it reads the information from the NFC module via I2C to determine what it sends to AWS and what response to send to the lock hardware package.

The NFC Shield module is a standalone NFC tag and reader which operates within the standard 4cm distance range, as is typical for NFC-based interactions. In iDoorlock, this will be the main hardware tag which will detect a nearby Android device with NFC enabled. This NFC Shield module will be wired to our Raspberry Pi, and the Raspberry Pi will use the PN532 NFC library to interact with the NFC Shield Module. The Raspberry Pi will be constantly polling to see if the NFC Shield module has detected a nearby NFC tag. Once a nearby NFC tag has been detected, the Raspberry Pi will send a series of messages including a message requesting an application and a message requesting a read for a pin number (which is the phone ID explained later). After the phone ID is received, the phone ID along with the lock ID associated with the lock is sent to AWS for authentication and validation in the form of an HTTPS GET request. The response indicates success/failure, and after receiving a response from the server or after timing out without receiving a response, the Raspberry Pi will then determine the appropriate response action; upon receiving verification of the user's information, the Raspberry Pi will power the servo motor for a set duration until it turns the deadbolt open. The green front facing indicator LED will flash three times to indicate this. The lock will then automatically relock itself after five seconds. Alternatively, there is also a manual lock button implemented into the side of the design. If authentication fails, a red front facing indicator LED will turn on for two seconds, and the lock will not open. At any time if the NFC shield fails or the connection is interrupted, nothing will be sent to the server and the lock state will be unchanged.

The entire hardware system is placed into a compact 3D printed housing, made in Autodesk Inventor and 3D printed with the resources available at TechSpark. It features cutouts for all the various components of the door lock and is assembled by placing the Raspberry Pi at the bottom of the housing and then inserting the deadbolt, the servo motor and 3D printed connector, and the lock cylinder in that order. Finally, a portable power bank can be placed on top of the other components, which will power the Raspberry Pi and fit well under the top of the housing. Assembling it with a tight fit allows for a relatively compact design in which we can house all of the electronics and that could be convincingly placed at a door instead of a traditional deadbolt. There are a few holes for debugging, but those would be removed for final project creation.

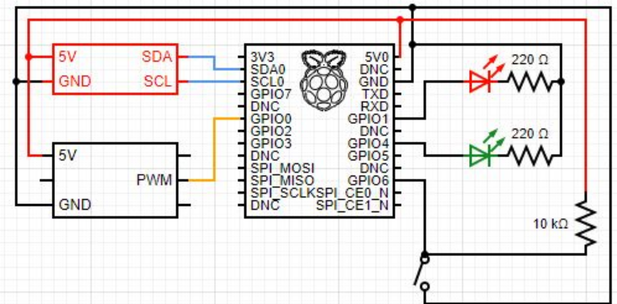


Fig. 4. Circuit diagram for iDoorlock. The top left red module is the NFC Shield Module, the bottom left module is the servo motor, and on the right there are the red and green LEDs as well as the manual deadbolt lock button, represented by the switch.

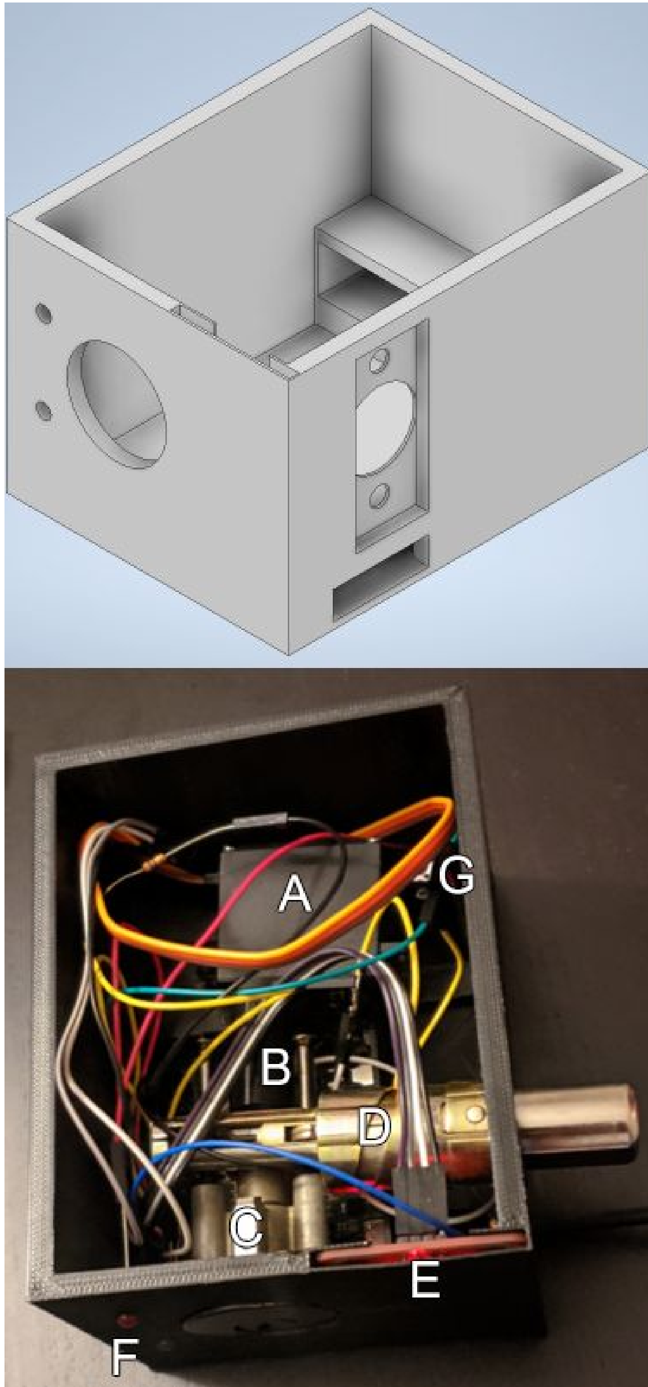


Fig. 5. Top: CAD design of iDoorlock housing. Bottom: completed iDoorlock working prototype. A: MG995 servo motor. B: Lock cylinder tailpiece-servo motor connector. C: Lock cylinder. D: Deadbolt. E: NFC Shield module. F: Indicator LEDs. G: Manual lock button.

B. AWS

iDoorlock uses AWS for central data processing and user permissions management. Any requests made from either the phone application or the Raspberry Pi are made to a REST API set up in API Gateway. The Raspberry Pi will send requests to the API when a user has touched their phone to the NFC terminal, and it will send the NFC information of the phone and the NFC reader. The phone application will send requests to the API when a user is registering a new lock or adding other users to the permissions list for a registered lock.

The Raspberry Pi and the phone application will assume an IAM role that only has permissions to invoke the API and the appropriate Lambdas. This is to ensure that malicious requests are not able to compromise the system. When either of them make a request to the API, they send their IAM access keys, which are then authenticated by the API. If the access keys are valid and have IAM permissions to invoke the Lambda and the API, then the API will forward the request to the Lambda. Otherwise, the request is rejected.

Our DynamoDB will be configured to have five different tables. The first table contains the allowed users for each lock. Each item will have a primary key that is the NFC information of the lock and another attribute that is a list of the NFC information of the phones that are allowed to unlock the lock. The second table contains the human-readable ID for each phone app. Each item will have a primary key of the NFC information and another attribute that is the name of the user that is passed in at registration time. The third table is similar to the third table except that it contains the human-readable ID of each lock. The fourth table contains the unlocking history of each lock. Each item will have a primary key that is the NFC info of the lock and other attributes for timestamp, user ID, and whether the request succeeded or failed. The fifth and final table contains the IAM keys that are expected to be passed in by the phone application and Raspberry Pi. Each item will have a primary key that is the public IAM key and another attribute for the secondary key.

On new lock registration, the phone application will send the user ID and a lock name that a user enters, as well as a pre-generated lock ID that uniquely identifies the Raspberry Pi. The Lambda will take the lock ID and enter it into the DynamoDB table that stores all locks. If the user registering the lock is new, then the Lambda will also add the user information to the corresponding table and add it to the permissions list for the lock. If the user already exists, the Lambda does not add new user information to DynamoDB and the server sends a response to the phone stating that the user has already been registered.

When the user removes a lock from their phone application, the phone application will send a request to API Gateway containing the phone ID and the lock ID to be removed. The Lambda will then check the DynamoDB table containing the list of phones allowed to open each lock and retrieve the item corresponding to the provided lock ID. If the phone ID is present in that item, then it will be removed. The server will send a response to the phone letting it know that it has been removed. Otherwise, no action is taken on the backend and the server sends a response to the phone letting it know that it is already removed.

When the user changes their username on the phone application, the phone application will send a request to API Gateway containing the phone ID and the updated username. Lambda will then check the DynamoDB table containing the list of registered phone IDs. If the phone ID

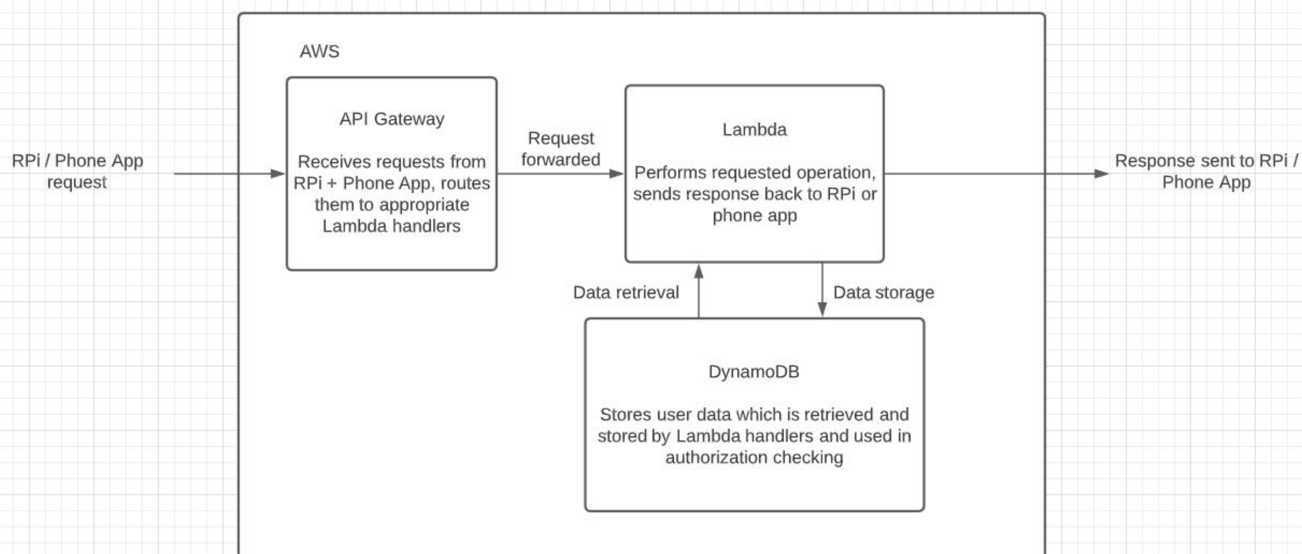


Fig. 6. AWS architecture diagram

does not exist, Lambda returns an error response to the phone application. If it does exist, Lambda will update the username attribute of the item and send a success response to the phone application.

When the user opens the lock history screen for a lock, the phone application sends a request to API Gateway containing the lock ID that the user wishes to see the lock history for and the user's phone ID. Lambda will then check DynamoDB to see if the provided phone ID has permission to unlock the provided lock ID. If it does not, then Lambda will return an error response to the phone application. Otherwise, Lambda will retrieve the lock history from DynamoDB for the provided lock and send a success response to the phone containing the requested lock history.

When the lock is set up and a user touches their Android phone to the lock, the NFC reader sends the NFC information of the phone and the reader to the Raspberry Pi. The Raspberry Pi then sends that information in a request to the API. The Lambda receives the NFC information and queries DynamoDB to check that the user has permission to access the lock. If the user has the correct permissions, then the Lambda will send a success response to the Raspberry Pi so it knows to open the lock. If the user does not have the correct permissions, then the Lambda sends a failure response to the Raspberry Pi so that the door will remain locked. In both cases, it will add an entry to the unlock history list for the lock with the user ID, timestamp, and whether or not the request succeeded.

C. Android Phone Application

The phone application will act as a key and have some interface options for controlling the locks. Using the tag dispatch system, touching the phone to the NFC shield will enable the phone to start a service that correctly verifies the sender is using the right protocol and application. Then after a request message is verified, the phone ID will be sent to the NFC for verification. The messages are called APDU (Application Protocol Data Units), and are used for NFC communication. The service is also set to require the device to be unlocked, so that even if someone not authorized to unlock the lock has the phone, he/she would not be authorized to unlock the lock. Due to the ubiquity of NFC technology, we will be able to reasonably support any Android device with NFC capabilities. Behind the scenes, the NFC shield will receive the phone ID and send the phone ID and lock ID to the server for authentication.

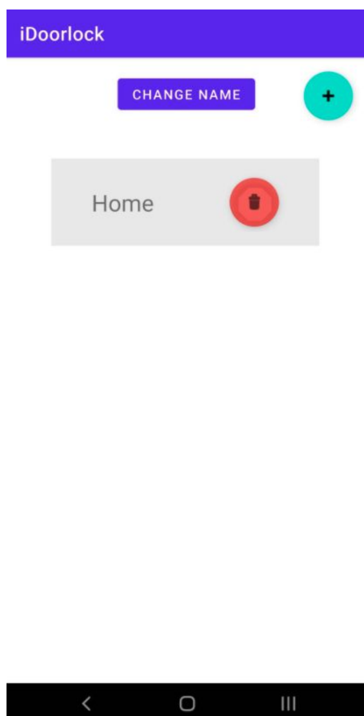


Fig. 7. Phone application home screen

The phone ID mentioned previously is a separate entity to identify each user. Upon first opening the app, the user will be told to register a name. The name is then sent via HTTPS to the server, and the server responds with a new and unique phone ID. The name that is sent will be associated with the phone ID on the server side for human readability purposes (identifying users with a multi digit ID isn't practical).

The app also has features to register locks, and with each unique code for a lock, the app will send information to the server for registration, specifically a lock ID and a phone ID. The lock ID will then associate the phone ID with the lock ID, and on uses of the lock, the server will be able to check this information against its stored information from the time of registration. The lock name is entered so the user is able to identify locks easily instead of using a multi digit lock ID. The lock name will be visible to the user upon registration at the home screen. The user can also delete saved locks after registration (which will also inform the server that the lock information saved for that particular phone is no longer valid).

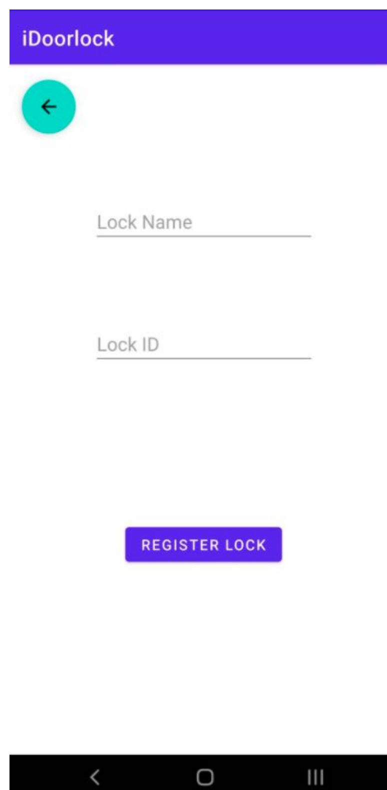


Fig. 8. Phone application lock registration screen

Another feature is that on the home screen, the user can choose certain locks to view information about the lock such as previous unlocks, time of unlock, and name of user that unlocked.

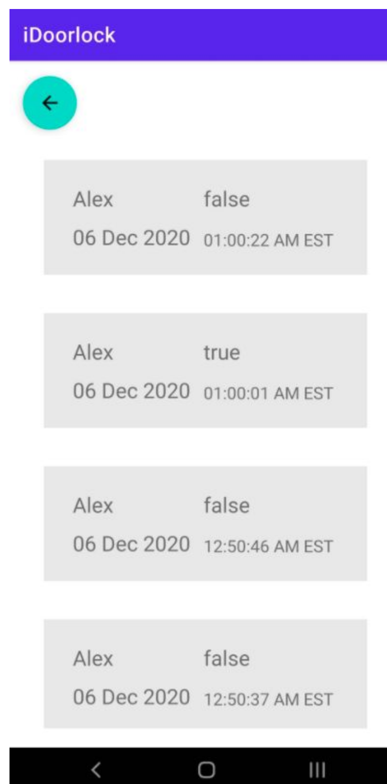


Fig. 9. Phone application lock history screen

VI. Project Management

A. Schedule

Our work was distributed into sections of around 1 to 2 week periods of development, with ample time at the end for integration/slack. We revised our schedule after Thanksgiving break, as we used the slack time for printing out the functional motor horn and component housing. Due to errors in Techspark's 3D printing process, we had to do dozens of prints to get the proper sizing for the two parts we needed.

Fig. 10. Schedule marking planned work done by week (full version on page 10)

B. Team Member Responsibilities

Alex Li was responsible for the work with the Phone App, all of its functionality, and how it interacts with the server and the NFC reader. Alex Xu assembled the motor/lock mechanism and developed the Raspberry Pi scripts that triggered the LEDs, motors, and button, as well as designed and printed the component housing and servo motor cylinder. Michael Chen worked on setting up a web server using AWS services (Lambda, DynamoDB).

C. Budget

Overall we have a pretty standard parts list consisting of only the parts we need except for the Raspberry Pi 4's, which we ordered extras for to account for board failure and concurrent development. The AWS and circuit components are free (circuit components free with TechSpark lab), and the Android Phone's price varies depending on the model (we used our personal Android phones).

Part Name	Price
AWS	Free
Servo motor	$18.69
Deadbolt lock (may not need)	$12.54
NFC Reader	$9.69
Raspberry Pi 4 4GB Starter Kit (2)	$199.98
Rapsberry Pi 4 4 GB Board Only	$56.00
General Circuit Components	Free
Android Phone	Price varies

Fig. 11. Projected budget. (full version on page 11)

D. Risk Management

We planned on managing our risk by spending more time during our design phase so we had a well-defined set of goals for development. However, the biggest roadblock we did not foresee was the 3D printing of the component housing and servo motor horn cylinder, because Techspark was closed for a duration due to a COVID-19 case. In addition, we also encountered errors where 3D prints we made came out inaccurately, more frequently warped or visibly shifted off its base beyond usability in a manner that Techspark employees hadn't seen before; we predict that we lost around 10 days worth of time printing and reprinting parts. Luckily, we did end up getting a few prints that came out relatively error-free, and capstone TA Mobolaji helped out with his personal 3D printer to get us accurate parts concurrently with Techspark's labs. We were able to ramp up physical prototyping very quickly after our design document, and Alex Xu figured out how to print everything we would need, successfully mitigating the risk from the skillset side of CAD-related tasks.

Another possible risk we predicted would be the lock speed, which we wanted to to beat the speed of locks with keys. This could be a challenge for several factors, such as Wi-Fi latency or slow authentication software. To mitigate this, we tested both the server response latency and the lock turning speed in tandem to ensure that the time it took for the whole system to go from NFC tag detection to the actual deadbolt unlocking was just under two seconds. Few additional optimizations were needed upon our initial attempt to write efficient code, and we attribute part of our success with unlocking speed due to the removal of the additional Arduino Uno microcontroller we no longer needed.

A last challenge we needed to face was the battery of the device. If we wanted to reduce wireless latency, we would always need to maintain the connection from the Raspberry Pi to the server, but that would use a lot of power. If we did not maintain the connection, then we could save a lot of energy and keep up our system for a long time before battery replacement. Part of our solution was in the switch from websockets to REST API, which removes the need for a persistent connection being open, which would result in a slower power drain by the Raspberry Pi. We also learned that it could be powered by a simple phone battery pack, with the specific model we were using predicted to last a little over a week in time, but throughout the demonstration we kept it powered by a wall outlet.

VII. Related Work

The idea of a smart door lock is not unique to our project. A relevant example is CBORD, which provides Carnegie Mellon with card scanners that manage building and residential hall access. Students, staff and faculty touch their school ID cards to the scanner, which then unlocks the door if the person has the correct credentials. However, this approach is directed towards larger organizations, as there is considerable overhead of producing ID cards for a small number of people. iDoorlock is designed to be more accessible to regular consumers by using a phone app instead of ID cards.

More consumer-facing solutions for smart door locks already exist on the market. The August Wi-Fi Smart Lock and Google Nest are examples of smart key-free locks that allow users to control them through their phones. Users can provide temporary guest passcodes for their friends and family to access the lock. They can also remotely lock and unlock through the phone app. These solutions are heavily reliant on the security of the phone app, and a proficient hacker could likely compromise the application.

iDoorlock distinguishes itself from other existing consumer-focused solutions with the inclusion of an NFC reader. By requiring biometric or PIN authentication through the existing phone software as well as the NFC tag, we implement a form of two-factor authentication that is incredibly hard to spoof. Even with a highly knowledgeable hacker, they must be physically close to the NFC reader. If somebody really wanted to enter your house at that point, they would likely just break the door down.

VIII. Summary

Our current design is a thorough and innovative approach to the current locking market. We expect to meet our core requirements of speed and security because the underlying technology we expect to use has already been tested and proven for its reliability. Using a 3 part approach with a server, Android phone, and the locking apparatus, the design will quickly and correctly verify users with a simple tap of the phone. We hope to refine the approach so that it can compete and improve upon modern locking designs.

References

[1] Allison, C. (2019, March 13). How does NFC payment work? Retrieved October 20, 2020, from https://fin.plaid.com/articles/how-does-nfc-payment-work/

[2] Build your own NFC reader. (2018, January 08). Retrieved October 20, 2020, from https://www.themobileknowledge.com/knowledge-base/nfc-reader-design/

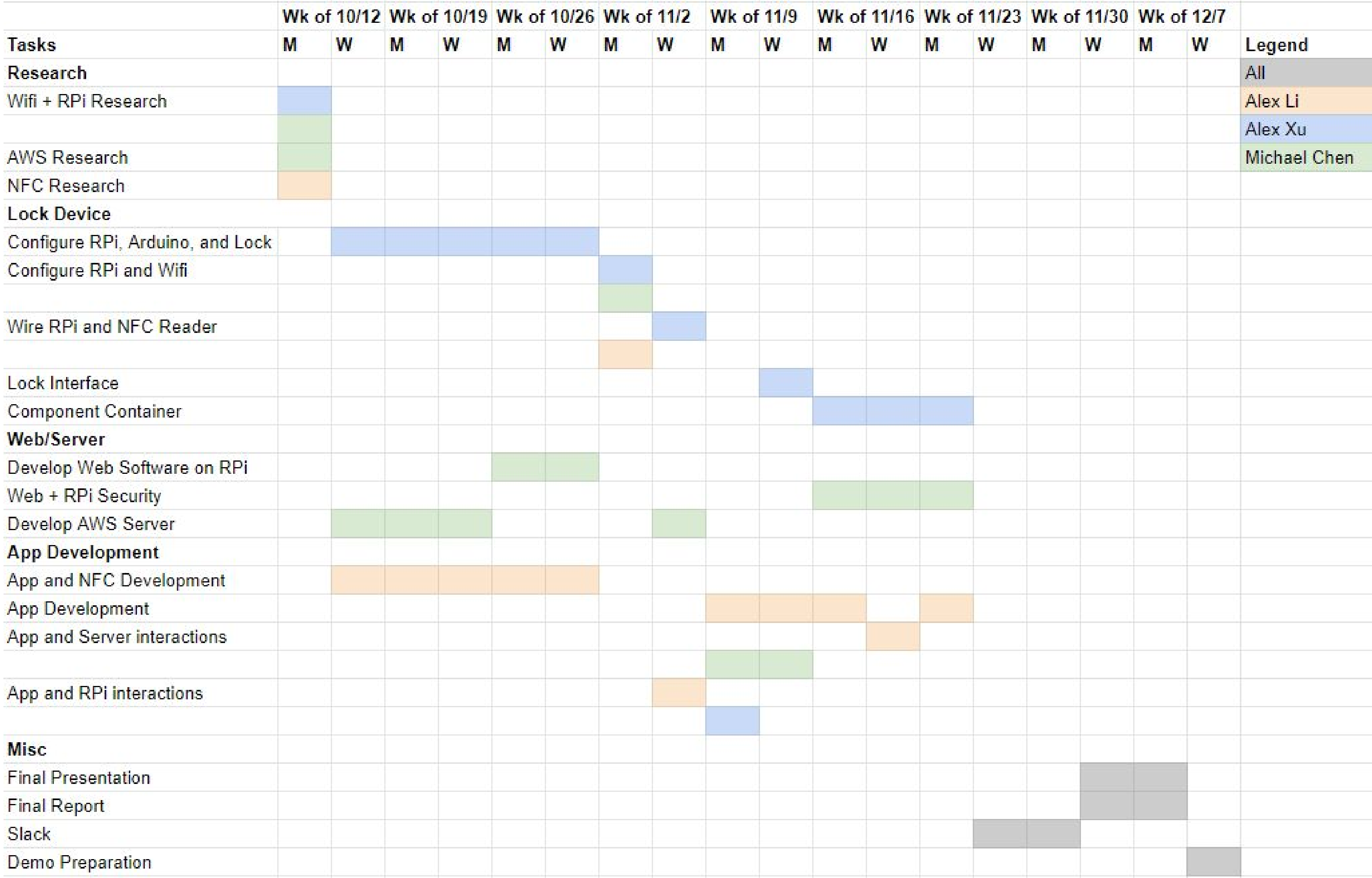
[3] NFC basics : Android Developers. (n.d.). Retrieved October 20, 2020, from https://developer.android.com/guide/topics/connectivity/nfc/nfc

[4] Square. (n.d.). How to Accept Apple Pay at Your Small Business. Retrieved October 20, 2020, from https://squareup.com/us/en/townsquare/apple-pay-for-small-businesses-how-to-accept-it

[5] Zuo, B. (n.d.). NFC Shield V2.0. Retrieved October 20, 2020, from https://wiki.seeedstudio.com/NFC_Shield_V2.0/

[6] Moore, Nicole Casal. "Hacking into Homes: Security Flaws Found in SmartThings Connected Home System." Michigan Engineering, University of Michigan, 2 May 2016, news.engin.umich.edu/2016/05/hacking-into-homes-security-flaws-found-in-smartthings-connected-home-system/.

[7] Monk, Simon. "Adafruit's Raspberry Pi Lesson 8. Using a Servo Motor." Adafruit Learning System, learn.adafruit.com/adafruits-raspberry-pi-lesson-8-using-a-servo-motor.


Wk of 10/12
Wk of 10/19
Wk of 10/26
Wk of 11/2
Wk of 11/9
Wk of 11/16
Wk of 11/23
Wk of 11/30
Wk of 12/7
Tasks
M
W
Legend
Research
All
Wifi + RPi Research
Alex Li
Alex Xu
AWS Research
Michael Chen
NFC Research
Lock Device
Configure RPi, Arduino, and Lock
Configure RPi and Wifi
Wire RPi and NFC Reader
Lock Interface
Component Container
Web/Server
Develop Web Software on RPi
Web + RPi Security
Develop AWS Server
App Development
App and NFC Development
App Development
App and Server interactions
App and RPi interactions
Misc
Final Presentation
Final Report
Slack
Demo Preparation

Part Name	Price	Source
AWS	Free	https://aws.amazon.com/
Servo motor	$18.69	https://www.amazon.com/Smraza-Helicopter-Airplane-Control-Arduino/dp/B07L2SF3R4/ref=sr_1_8?dchild=1&keywords=servo+motor&qid=1600987579&sr=8-8
Deadbolt lock (may not need)	$12.54	https://www.amazon.com/dp/B07J4VQCL7/ref=twister_B07PWWP91P?_encoding=UTF8&psc=1
NFC Reader	$9.69	https://www.amazon.com/HiLetgo-Communication-Arduino-Raspberry-Android/dp/B01I1J17LC/ref=sr_1_2?dchild=1&keywords=hiletgo+nfc+reader&qid=1603158965&s=electronics&sr=1-2
Raspberry Pi 4 4GB Starter Kit (2)	$199.98	https://www.amazon.com/CanaKit-Raspberry-4GB-Starter-Kit/dp/B07V5JTMV9/ref=pd_rhf_ee_p_img_2?_encoding=UTF8&psc=1&refRID=1FHGT3SEPY10HRNJBVYH
Rapsberry Pi 4 4 GB Board Only	$56.00	https://www.amazon.com/Raspberry-Model-2019-Quad-Bluetooth/dp/B07TC2BK1X/ref=pd_rhf_ee_p_img_1?_encoding=UTF8&psc=1&refRID=BFHF5ZCCN0S1XEE7M9J4
General Circuit Components	Free	TechSpark Lab
Android Phone	Price varies	N/A (Personal phone)