Group Members:
Dale Buencillo (dbuen4, dbuen4@uic.edu), Daniel Dedic (ddedi2, ddedi2@uic.edu), Khi Kidman (mkidma2, mkidma2@uic.edu)

Group Number: 18

Lock Buddy

Abstract

Lock Buddy is a door security system with 2 forms of authentication. An Arudino is used to take input, authenticate, and detect a user. Another opens the locking mechanism of the door, while a 3rd sends information to the owner's device. The originality of the product revolves around the door handle, which permits the users multiple ways to unlock the door while giving the owner a reliable method to know who has interacted with the door.

Description of Project Idea

For our group project, we are creating a door security authentication system with a focus on the door handle. The overall idea would be to use some form of user identification (fingerprint, NFC, etc.) and depending if the user is authorized to unlock the door, another device would be responsible for unlocking the door or keeping it locked. Additionally, a notification system would be put in place to notify the owner of the lock that someone attempted to unlock the door, sending the notification through wifi to the owner's selected device.

Initial Project Design

This project will need the use of at least 3 Arduinos. The first one will solely be responsible for handling the authentication of the user attempting to unlock the door. Making use of an **NFC/RFID** reader and a **fingerprint reader**, the user has two options to attempt to unlock the door. No matter the option and the result, a signal is sent to the 2nd and 3rd Arduinos which will act accordingly depending on the signal. The (metal) doorknob will also have capacitive touch functionality, sending a signal to the 2nd and 3d Arduinos if the doorknob is being touched.

The second Arduino will be focusing on controllering the locking mechanism of the door. In order to do this, the Arduino will need a **servo motor** that will physically turn the locking mechanism (determined to be a deadbolt lock) open if the correct fingerprint or RFID is

received. Additionally, an **LCD** and **LEDS** will be incorporated into the locking mechanism to let the user know whether or not the door has been unlocked.

Finally, the last Arduino will be focused on transmitting info about the door's activity to the owner of the door. Using the Arduino's built in wifi transmitter, the Arduino will send a signal to the owner, letting them know that someone is attempting to open the door if the RFID or fingerprint isn't recognized. Otherwise, if recognized, the device will notify the owner who opened the door. Additionally, a **buzzer** will be added to audibly respond to any attempts to open the door in tandem to notifying the owner and an **ultrasonic sensor** will also be included to be able to detect any persons who come in close proximity to the door and record/notify the owner of their presence.

Communication Between the Multiple Arduinos

The form of communication that will be utilized by the Arduinos will be using WiFi to communicate with an external database/server to notify the user of any door attempts, and using Serial between the Arduinos to send input data on door attempts and other input devices. Communications between Arduino 2 and 3 will be via Hardware Serial while communications between 1 and 3 will be via Software Serial. This allows multiple communication methods on Arduino 2 to be handled at the same time.

Original Work

The originality of this project revolves around the focus of the **door handle**. Many home security systems rely on just a sensor or visual indicator to let an owner know that someone is at or has opened the door. By relying on when an individual specifically has interacted with the door handle, the owner knows that someone is **actively attempting** to open the door rather than camera motion being detected, which could just be someone walking their dog past the door rather than a suspicious individual.

Additionally, the variety of secure authentication methods gives users multiple options to open the door, allowing them to open it even when they forget the appropriate device/tool needed to unlock the door.

Inputs and Outputs

Inputs	Outputs
NFC/RFID Reader (Form of Authentication, used to verify users to unlock the door)	LCD (Used to Indicate if Door is Open or Closed)
Fingerprint Reader (2nd Form of Authentication, used to verify users to unlock the door)	LEDs (Used to Indicate if Door is Open or Closed)
Ultrasonic Sensor (Detects movement close to the door, notifies owner of movement)	Buzzer (Alarm, notifies owner audibly if unauthorized users attempt to open the door multiple times)
Doorknob (When touched, sends owner notification that someone is interacting with door)	Servo Motor (Responsible for Moving Deadbolt/Locking Mechanism)

Completed Project Timeline from Start of Project to Finish

1/31 - 3/14	Project Ideas Discussion and Finalization (By 2/14) ● **Milestone 1 - 3**
3/3 - 3/7	Distribution of Work + Gathering Needed Materials
3/10 - 3/14	Working on Design Presentation
3/17 - 3/21	**Design Presentation** ● **Milestone 4-6**
3/24 - 3/28	Individual Work (Getting designated parts working)
3/31 - 4/4	More time to finalize individual work (Break) ● **Finalizing Fingerprint Sensor** ● **Combine Arduino 1 code sketches into one individual sketch** ● **Design Document Update (4/4, Milestone 7)**

4/7 - 4/11	**Initial Test Demo with All Working Parts** + Adjustments
4/14 - 4/18	**More Test Demo with All Working Parts** + Adjustments
4/21 - 4/25	Preparing for Project Demo Presentation
4/28	**Project Demo** ● **Milestone 8**
4/30	**Final Design Document** ● **Milestone 9**

Materials:

x3 Arduino R4 Wifi

x3 Breadboards

Resistors

Wires

x1 NFC/RFID Reader

x1 NFC/RFID Tag

x1 DY50 Fingerprint Sensor

x1 Deadbolt

x1 Servo Motor

x1 LCD I2C 4-pin

x2 LEDs (red and green)

x1 Button

x1 Ultrasonic Sensor

x1 Active Buzzer

References:

SoftwareSerial

Arduino 1 (Doorknob, etc)

https://www.instructables.com/Arduino-Wiring-and-Programming-of-RFID-Sensor/

https://randomnerdtutorials.com/fingerprint-sensor-module-with-arduino/

Arduino 2 (Servo, etc)

https://docs.arduino.cc/learn/electronics/servo-motors/

https://projecthub.arduino.cc/arduino_uno_guy/i2c-liquid-crystal-displays-5eb615

https://docs.arduino.cc/built-in-examples/basics/Blink/

Arduino 3 (WiFi Notification, etc)

projecthub.arduino.cc/lucasfernando/ultrasonic-sensor-with-arduino-complete-guide-284faf

https://www.circuitbasics.com/how-to-use-active-and-passive-buzzers-on-the-arduino/

https://docs.arduino.cc/tutorials/uno-r4-wifi/wifi-examples/

https://supabase.com/docs/guides/api

https://supabase.com/docs/guides/functions/examples/push-notifications

Diagram of Arduino 1 (Doorknob, Fingerprint Scanner, RFID Reader; Linked to Arduino 2 and 3 separately via serial)

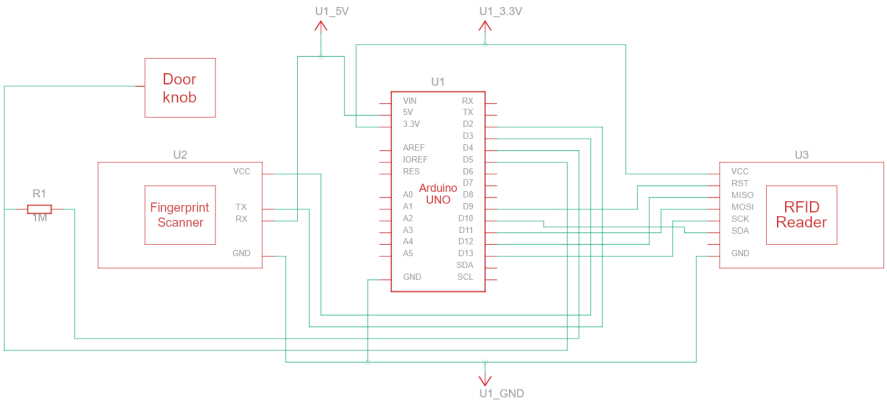


Diagram of Arduino 2 (Servo, LCD, Button and LEDs; Linked to Arduino 1 via Serial)

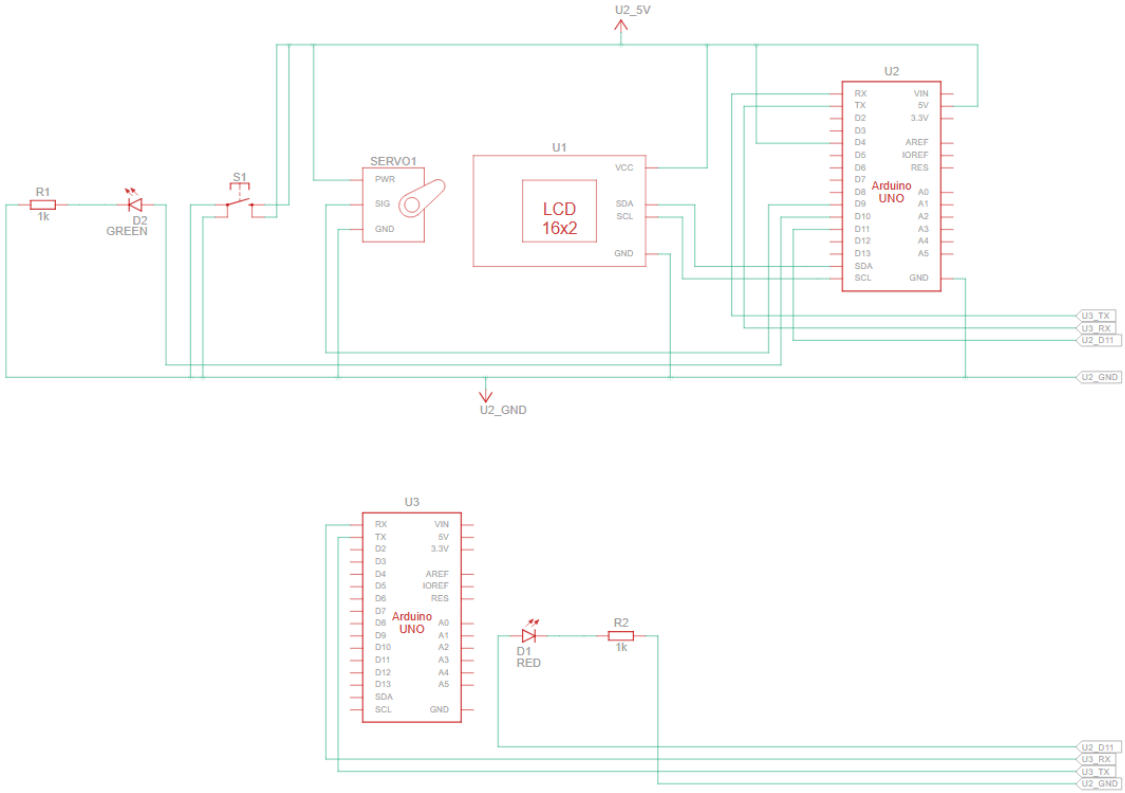
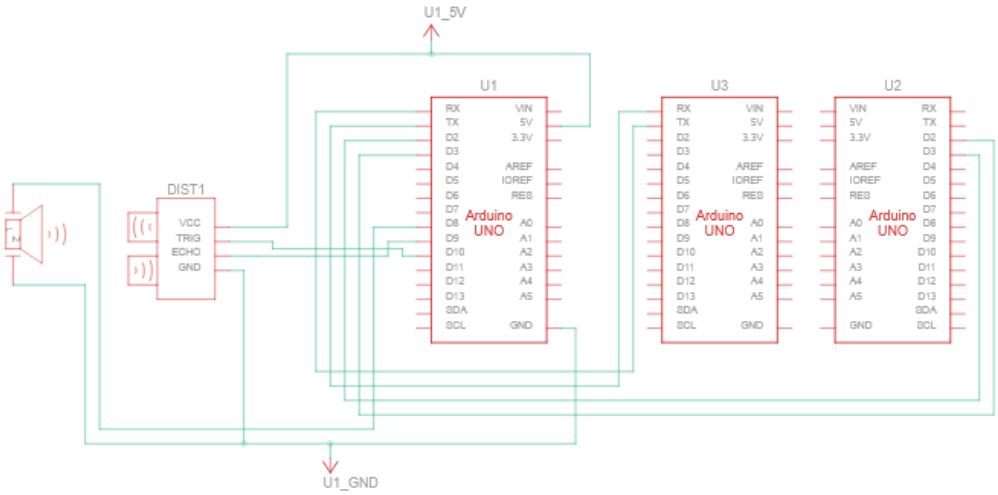


Diagram of Arduino 3 (Buzzer and Ultrasonic; linked to Arduino 1 via SoftwareSerial communication and Arduino 2 via Serial1)



How to Build

Arduino 1 Setup

1. Wire Arduino R4 to a breadboard; 5v pin to 5v line, 3v pin to second voltage line and GND pin to GND line
2. Connecting Capacitive Touch Circuit
    a. Pin 6 to a breadboard column
    b. Connect a 1M resistor from the pin 6 column to another adjacent column.
    c. Pin 7 to the same column as the opposite end of the resistor from the pin 6 connection.
    d. Using a wire, connect the doorknob to the column that connects with pin 7.
3. Connecting Fingerprint Scanner
    a. Connect pin 1 to fingerprint scanner TX
    b. Connect pin 0 to fingerprint scanner RX
    c. Connect breadboard 3v to fingerprint scanner VCC
    d. Connect GND from arduino or breadboard to fingerprint scanner GND
4. Connecting RFID Reader
    a. SDA to pin 10 on Arduino 1
    b. SCK to pin 13 on Arduino 1

    c. MOSI to pin 11 on Arduino 1
    d. MISO to pin 12 on Arduino 1
    e. IRQ is not used (leave unconnected)
    f. GND to ground line on breadboard
    g. RST to pin 9 on Arduino 1
    h. 3.3V (not 5V!) to 3.3V pin on Arduino 1 (5v may damage the RFID reader!)
5. Upload Arduino 1 Code

Arduino 2 Setup

1. Wire Arduino R4 to a breadboard; 5v pin to 5v line (+) and GND pin to GND line (-)
2. Connecting Button
    a. PIN 4 to same column as Upper Right section of button
    b. Connect Button Left section to GND (-)
    c. Connect Bottom Right section to 5v line (+)
3. Connecting I2C LCD (Female to Male Jumper Wires Recommended)
    a. GND to GND Line on Breadboard
    b. VCC to 5v Line on Breadboard
    c. SDA to SDA on UNO R4
    d. SCL to SCL on UNO R4
4. Connecting Red Led
    a. Pin 10 to same column as Anode (Long Leg)
    b. Connect a 220 Ohm Resistor, one leg in same column as Cathode (Short Leg) and other leg in GND line
5. Connecting Green Led
    a. Pin 10 to same column as Anode (Long Leg)
    b. Connect a 220 Ohm Resistor, one leg in same column as Cathode (Short Leg) and other leg in GND line
6. Connecting Servo
    a. Connect PIN 9 to CTRL (Usually Orange Wire)
    b. Connect GND line to GND (Usually Brown Wire)
    c. Connect 5v line to VCC (Usually Red Wire)
7. Uploaded Arduino 2 code

a. Ensure you have the Servo.h and LiquidCrystal_I2C.h libraries installed.

Arduino 3 Setup

1. Wire Arduino R4 to a breadboard; 5v pin to 5v line and GND pin to GND line.
2. Connect the Ultrasonic Sensor to the R4 (place on breadboard and connect)
    a. VCC to the 5v line
    b. TRIG to pin 10
    c. ECHO to pin 9
    d. GND to ground line
3. Connect the Passive Buzzer to the R4 (place on breadboard and connect)
    a. Positive end to pin 8
    b. Negative end to ground line
4. Initial circuit setup is completed; upload Arduino 3 code to Arduino.
    a. Insure you have secrets.h setup in same directory, with defines for SECRET_SSID_HOME (wifi name), SECRET_PASS_HOME (wifi password), and SUPABASE_API_ANON (anon key for supabase database)
5. *If you wish to setup your own database for push notifications, setup a Supabase Project to get the SUPABASE_API_ANON and the URL (update in arduino code accordingly)*
    a. *Setup tables for 'log': the post events, profiles: user ID linked with the FCM token for push notification, and 'test': userID linked to full name.*
    b. *Setup the push notification function, more detailed instructions found here: https://supabase.com/docs/guides/functions/examples/push-notifications?queryGroups=platform&platform=fcm*
    c. *Get the app source code and build in Android Studio, updating your URL and keys accordingly with a created secret.java class (where String secret = "your anon key"). https://github.com/DGC9Crew/lock-buddy-app/tree/master*

Serial Connection Setup

1. Connect serial wiring (software serial) from Arduino 1 to Arduino 3
    a. Arduino 1 pin 4 to Arduino 3 pin 3 (input)
    b. Arduino 1 pin 5 to Arduino3 pin 2 (output)
    c. Connect both Arduino's GND to share ground

2. Connect serial wiring (serial1) from Arduino 2 to Arduino 3
    a. Arduino 1 pin TX to Arduino 3 pin RX (input)
    b. Arduino 1 pin RX to Arduino TX pin 0 (output)
    c. Connect both Arduino's GND to share ground

Note: Arduino's can be connected to a shared GND using a 4th breadboard, where each GND is connected to the same GND line

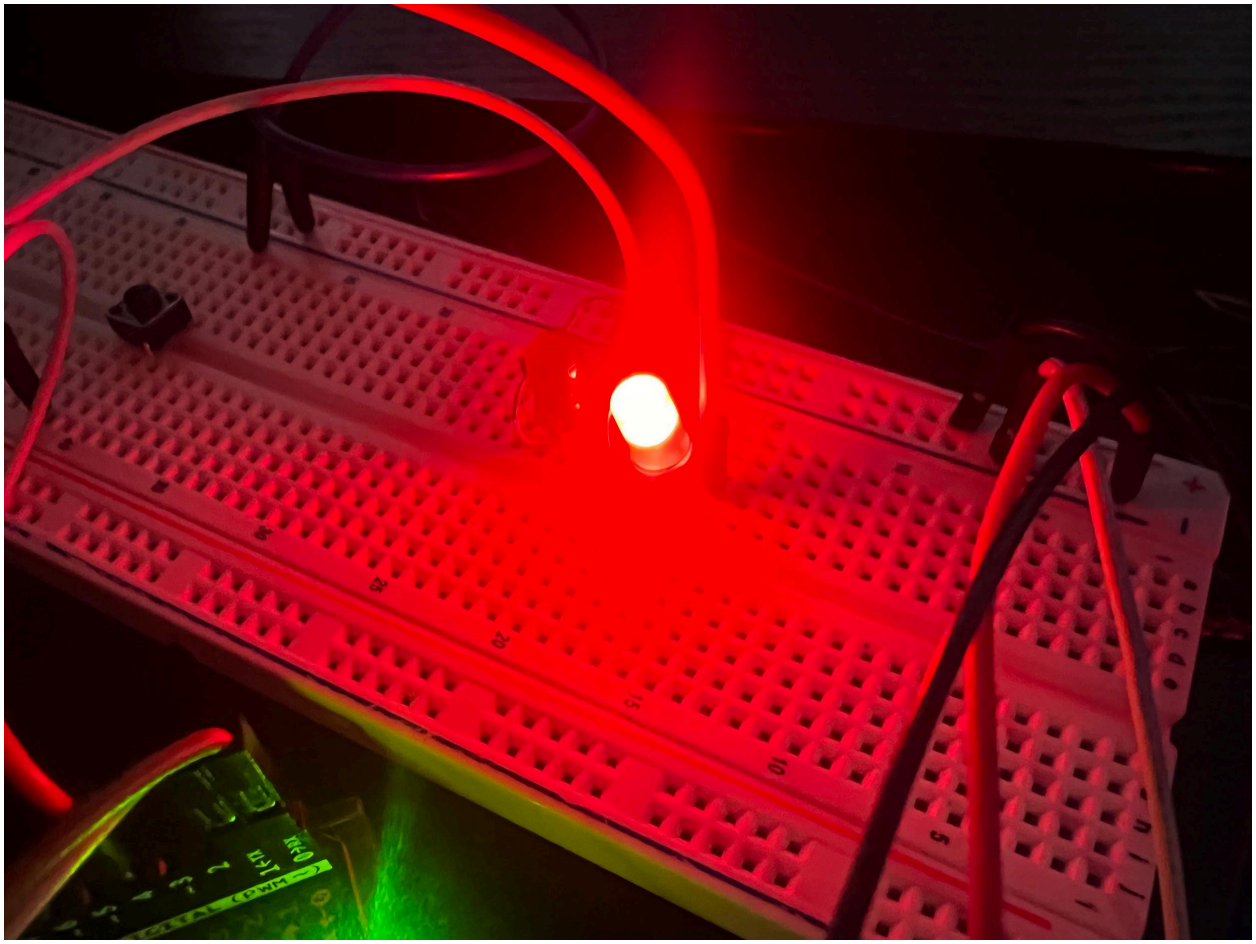
User Guide

- Attach the RFID and Fingerprint Reader on the side of the door facing the outside within close proximity of the door handle (so guest may interact with it)
- Attach the Ultrasonic Sensor so that it's facing AWAY from the door
- On the other side of the door, attach the 3 Arduinos within close proximity of a power supply
    - Place Arduino 2 as close as possible to the deadbolt responsible for the locking mechanism
        - Attach the servo motor to the deadbolt mechanism, allowing it to move left and right appropriately.
- Setup database and wifi info as seen in Arduino 3 setup
- **Your lock buddy should be ready to go!**

Examples of Functioning I/O Devices

I2C LCD

LEDs



Note: (Other devices have no still-image visual indicator, but can be observed moving or sending messages to the terminal)

Code Segments:

ARDUINO 1 CODE

```
#include <SPI.h>
#include <MFRC522.h>
#include <SoftwareSerial.h>
#include <Adafruit_Fingerprint.h>

const uint8_t RST_PIN = 9;
const uint8_t SDA_PIN = 10;
```

```
// SoftwareSerial arduino2(2, 3); // removed due to UNO R4 limitations
SoftwareSerial arduino3(4, 5);


MFRC522 rfidReader(SDA_PIN, RST_PIN);

#define mySerial Serial1

Adafruit_Fingerprint fingerprintScanner = Adafruit_Fingerprint(&mySerial);

int capacativeTouchSendPin = 6;  // Pin used to charge the sensor
int capacativeTouchSensePin = 7; // Pin used to read capacitance
long baselineCapacitance = 0; // Stores the baseline capacitance value

int ledPin = 8;

unsigned long previousTouchMillis = 0;
unsigned long capacativeTouchDetectDelayMillis = 3000;

unsigned long previousRFIDReadMillis = 0;
unsigned long RFIDReadDelayMillis = 2000;

unsigned long previousFingerprintAttemptMillis = 0;
unsigned long fingerprintAttemptDelay = 3000;

String uidString;
int key[] = {131,161,38,20};
int keyRead = 0;

struct User {
```

```
  char name[8];
  int key[4];
  uint8_t id;
};

struct User admin = {"Admin", {131,161,38,20}, 1};
struct User user69 = {"User 69", {0, 0, 0, 0}, 69};
struct User dan = {"Dan", {4, 87, 26, 194}, 42};

struct User users[] = {admin, user69, dan};

bool readRFID() {
   rfidReader.PICC_ReadCardSerial();
   //Serial.println("Scanned PICC's UID:"); // debug purposes
   //printDec(rfidReader.uid.uidByte, rfidReader.uid.size); // debug purposes

   uidString = String(rfidReader.uid.uidByte[0])+" "+String(rfidReader.uid.uidByte[1])+"
"+String(rfidReader.uid.uidByte[2])+" "+String(rfidReader.uid.uidByte[3]);

   //Serial.print("UID: "); // debug purposes
   //Serial.print(uidString); // debug purposes

   int i = 0;
   bool match = true;
   while (i < rfidReader.uid.size) {
      if (rfidReader.uid.uidByte[i] != key[i]) {
         match = false;
      }
      i++;
   }
```

```
    // Debug printing
    // if (match) {
    //     Serial.println("Key recognized");
    //     return true;
    // } else {
    //     Serial.println("Unknown key");
    //     return false;
    // }
}

void printDec(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? " 0" : " ");
        Serial.print(buffer[i], DEC);
    }
}

void calibrateCapacitiveTouch() {
    digitalWrite(capacativeTouchSendPin, LOW);

    // Calibration Step
    Serial.println("Calibrating... Do not touch the doorknob.");
    delay(2000); // Wait 2 seconds to ensure it's untouched

    long total = 0;
    const int samples = 20; // Number of calibration samples

    for (int i = 0; i < samples; i++) {
        total += measureCapacitance();
        delay(50); // Short delay between measurements
    }
```

```
    baselineCapacitance = total / samples; // Calculate baseline average
    Serial.print("Calibration complete. Baseline: ");
    Serial.println(baselineCapacitance);
}

bool detectCapacativeTouch() {
    long capValue = measureCapacitance();
    long change = capValue - baselineCapacitance; // Detect change from baseline

    // Debug printing
    // if (change > 1) { // Adjust threshold as needed
    //     Serial.println("Touch detected!");
    // }
    //digitalWrite(ledPin, change > 3);
    return change > 2;
}

// Function to measure capacitance
long measureCapacitance() {
    pinMode(capacativeTouchSensePin, INPUT);
    digitalWrite(capacativeTouchSendPin, HIGH);
    delayMicroseconds(10);

    pinMode(capacativeTouchSensePin, OUTPUT);
    digitalWrite(capacativeTouchSensePin, LOW);

    pinMode(capacativeTouchSensePin, INPUT);
    return pulseIn(capacativeTouchSensePin, HIGH, 5000); // Measure charge decay time
}
```

```
// returns -1 if failed, otherwise returns ID #
int getFingerprintID() {
  uint8_t p = fingerprintScanner.getImage();
  if (p != FINGERPRINT_OK)  return -1;

  p = fingerprintScanner.image2Tz();
  if (p != FINGERPRINT_OK)  return -1;

  p = fingerprintScanner.fingerFastSearch();
  if (p == FINGERPRINT_NOTFOUND) {
    return 0;
  } else if (p != FINGERPRINT_OK) {
    return -1;
  }
  // found a match!
  return fingerprintScanner.fingerID;
}

void setup() {
    // arduino2.begin(9600); // removed due to UNO R4 limitations
    arduino3.begin(9600);
    Serial.begin(9600);
    pinMode(capacativeTouchSendPin, OUTPUT);
    pinMode(capacativeTouchSensePin, INPUT);
    pinMode(ledPin, OUTPUT);

    SPI.begin();
    rfidReader.PCD_Init();

    fingerprintScanner.begin(57600);
    delay(5);
```

```
  if (fingerprintScanner.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1) { delay(1); }
  }

  calibrateCapacitiveTouch();
}

void loop() {

  unsigned long currentMillis = millis();

  if (detectCapacativeTouch()) {
    if (currentMillis - previousTouchMillis > capacativeTouchDetectDelayMillis) {
      arduino3.print("{touch}");
    }
    digitalWrite(ledPin, HIGH);
    previousTouchMillis = currentMillis;
  }

  if (currentMillis - previousTouchMillis > capacativeTouchDetectDelayMillis) {
    digitalWrite(ledPin, LOW);
  }

  if (rfidReader.PICC_IsNewCardPresent()) {
    if (currentMillis - previousRFIDReadMillis > RFIDReadDelayMillis) {
      readRFID();
      Serial.println(uidString);
      bool validUserFound = false;
```

```
    for (User user : users) {
      bool isValid = true;
      for (int i = 0; i < 4; i++) {
        if (user.key[i] != rfidReader.uid.uidByte[i]) {
          isValid = false;
        }
      }
      if (isValid) {
        validUserFound = true;
        arduino3.print("{"); arduino3.print(user.name); arduino3.print("}");
        break;
      }
    }
    if (!validUserFound) {
      arduino3.print("{fail}");
    }
    previousRFIDReadMillis = currentMillis;
  }

}

if (currentMillis - previousFingerprintAttemptMillis > fingerprintAttemptDelay) {
  int fingerprintID = getFingerprintID();
  switch (fingerprintID) {
    case 0:
      Serial.println("Unauthorized fingerprint access");
      arduino3.print("{fail}");
      previousFingerprintAttemptMillis = currentMillis;
      break;
    case -1:
      break;
```

```
      default:
        Serial.print("Fingerprint access: ID #"); Serial.println(fingerprintID);
        for (User user : users) {
          if (user.id == fingerprintID) {
            arduino3.print("{"); arduino3.print(user.name); arduino3.print("}");
            // arduino2.print(user.name); // removed due to UNO R4 limitations
            Serial.print("Welcome "); Serial.println(user.name);
          }
        }
        previousFingerprintAttemptMillis = currentMillis;
        break;
    }
  }

  // if (currentMillis - previousRFIDReadMillis > RFIDReadDelayMillis) {
  //   if (rfidReader.PICC_IsNewCardPresent()) {
  //       if (readRFID()) {
  //           digitalWrite(ledPin, LOW);
  //       }
  //   }
  //   previousRFIDReadMillis = currentMillis;
  // }
}
```

ARDUINO 2 CODE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>

// define pins
#define SERVO_PIN 9
```

```
#define GREEN_LED 10
#define RED_LED 11
#define CLOSE_BUTTON_PIN 4


// initialize components
Servo doorLock;
LiquidCrystal_I2C lcd(0x27, 16, 2);



// state tracking
bool isUnlocked = false;
String receivedData = "";


// debounce variables
unsigned long lastButtonPress = 0;
const unsigned long debounceDelay = 200;


void setup() {
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(CLOSE_BUTTON_PIN, INPUT_PULLUP);


  digitalWrite(GREEN_LED, LOW);
  digitalWrite(RED_LED, HIGH);
```

```
  doorLock.attach(SERVO_PIN);
  doorLock.write(0);


  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Door Locked   ");
  lcd.setCursor(0, 1);
  lcd.print("               ");


  Serial1.begin(9600);
}


void loop() {
  unsigned long currentTime = millis();

  // check button press to close door
  if (digitalRead(CLOSE_BUTTON_PIN) == LOW && isUnlocked) {
    if (currentTime - lastButtonPress > debounceDelay) {
      lockDoor();
      lastButtonPress = currentTime;
    }
  }

  // check for button press to turn off alarm
  else if (digitalRead(CLOSE_BUTTON_PIN) == LOW) {
    if (currentTime - lastButtonPress > debounceDelay) {
      Serial1.print("F");
```

```
      lastButtonPress = currentTime;
    }
  }

  // Check for message from Arduino 1 to unlock the door
  if (Serial1.available()) {
    receivedData = Serial1.readStringUntil('\n');
    receivedData.trim();

    Serial.println(receivedData);

    if (receivedData.length() > 0 && !isUnlocked) {
      unlockDoor(receivedData);
    }
  }
}


//unlock function. displays user name based on message recieved
void unlockDoor(String message) {
  doorLock.write(90);
  digitalWrite(GREEN_LED, HIGH);
  digitalWrite(RED_LED, LOW);
  lcd.setCursor(0, 0);
  lcd.print("Door Unlocked ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print("Welcome " + message);
  isUnlocked = true;
```

```
  Serial.println("Door unlocked");
}


//lock function, resets after button is pressed
void lockDoor() {
  doorLock.write(0);
  digitalWrite(GREEN_LED, LOW);
  digitalWrite(RED_LED, HIGH);
  lcd.setCursor(0, 0);
  lcd.print("Door Locked   ");
  lcd.setCursor(0, 1);
  lcd.print("               ");
  isUnlocked = false;

  Serial.println("Door locked");
}
```

ARDUINO 3 CODE

```
// Team Number: 18
// Dale Buencillo (NetID: dbuen4), Daniel Dedic (NetID: ddedi2), Khi Kidman (NetID: mkima2)
// Project Name: Lock Buddy
// Abstract:      Lock Buddy is a door security system with 2 forms of authentication.
// An Arudino is used to take input, authenticate, and detect a user.
// Another opens the locking mechanism of the door, while a 3rd sends information to the
owner's device.
// The originality of the product revolves around the door handle, which permits the users
multiple ways to unlock the door while giving the owner a reliable method to know who has
interacted with the door.
```

```
#include "WiFiS3.h"
#include "WiFiSSLClient.h"
#include "IPAddress.h"
#include "secrets.h"
#include <SoftwareSerial.h>
char ssid[] = SECRET_SSID_HOME;
char pass[] = SECRET_PASS_HOME;
int status = WL_IDLE_STATUS;
char server[] = "bqgbsomhldpfsbgfkgsb.supabase.co";

WiFiSSLClient client;
const int trigPin = 10;
const int echoPin = 9;
const int buzzer = 8;
long duration;
int distance;
bool gathering = false;
char data[64] = {0};
int index=0;
unsigned long timer = 0;
unsigned long sonicReportTimer = 0;

unsigned long sonicPinTimer = 0;

unsigned long touchTimer = 0;
int sonicStage = 0;
int numberAttempts = 0;

unsigned long chirpTimer = 0;
int numberChirp = 0;
```

```
SoftwareSerial mySerial(2,3);

void checkReconnect() {
  if (!client.connected()) {
        Serial.print(".");
        client.connect(server, 443);
        }
}
void updateLockState(bool state) {
        checkReconnect();

        char nm[8] = {0};
        if(state) {
        strcpy(nm, "true");
        } else {
        strcpy(nm, "false");
        }
        client.println("PATCH /rest/v1/lockstate?select=locked&id=eq.1 HTTP/1.1");
        client.println("Host: bqgbsomhldpfsbgfkgsb.supabase.co");
        client.print("apikey: ");
        client.println(SUPABASE_API_ANON);
        client.println("Content-Type: application/json");
        client.println("Connection: keep-alive");
        client.print("Content-Length: ");
        client.println((strlen(nm)+11));
        client.println();
        client.print("{\"locked\":");

        client.print(nm);
        client.println("}");
```

```
        client.println();
}



void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzer,OUTPUT);
  Serial.begin(9600);
  mySerial.begin(9600);
  Serial1.begin(9600);
  while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
        Serial.println("Communication with WiFi module failed!");
        // don't continue
        while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
        Serial.println("Please upgrade the firmware");
  }

  // attempt to connect to WiFi network:
int attemptConnect = 0;
  while (status != WL_CONNECTED) {
```

```
        attemptConnect += 1;
        if( attemptConnect > 3) {
                exit(0); // end after 3 failed attempts to connect
        }
        Serial.print("Attempting to connect to SSID: ");
        Serial.println(ssid);
        // Connect to WPA/WPA2 network.
        status = WiFi.begin(ssid, pass);

        // wait 10 seconds for connection:
        // NOTE: This usage of delay is for waiting for the WiFi connection to begin! This usage
of delay will not have the program continue, allowed with
https://piazza.com/class/m5vcnhr6sb74je/post/446
        delay(10000);
  }

  printWifiStatus();
  client.connect(server, 443);
  timer = millis();
  sonicReportTimer = millis() + 60*1000;
  sonicPinTimer = millis();
  touchTimer = millis();
}
void reportEvent(String event) {
        client.println("POST /rest/v1/log HTTP/1.1");
        client.println("Host: bqgbsomhldpfsbgfkgsb.supabase.co");
        client.print("apikey: ");
        client.println(SUPABASE_API_ANON);
        client.println("Content-Type: application/json");
        client.print("Content-Length: ");
        client.print((15+event.length()));
```

```
            client.println();
            client.println();
            client.print("{\"eventID\": \"");
            client.print(event);
            client.print("\"");
            client.println("}");
            client.println();
}
void evaluateParams(char** a) {

   if(strcmp(*a, "touch") == 0) {
            if(millis() - touchTimer > 10 * 1000) {
            reportEvent("touch");
            touchTimer = millis();
            }
  } else if(strcmp(*a, "fail") == 0) {
            Serial.println("ERR TO GET IN");
            numberChirp=3;
            chirpTimer = millis();
            numberAttempts+=1;
            reportEvent("fail");
  } else if(strcmp(*a, "alarm") == 0) {
            digitalWrite(buzzer, HIGH);
  } else if(strcmp(*a, "silence") == 0) {
            Serial.println("silenced..");
            digitalWrite(buzzer, LOW);
  } else {
            Serial.println("ITS OPEN!");
            numberChirp=1;
            chirpTimer = millis();
            updateLockState(false);
```

```
        Serial1.println(*a);
        numberAttempts = 0;
        reportEvent(*a);
  }
}


void loop() {
  //read_response();
  if (Serial1.available()) {
        char b = Serial1.read();
        Serial.println(b);
        numberAttempts = 0;
        digitalWrite(buzzer, LOW);
  }
  if(numberAttempts >= 5) {
        digitalWrite(buzzer, HIGH);
  }
  if(millis() - timer > 5000) {
        client.println("GET /rest/v1/lockstate?select=locked&id=eq.1 HTTP/1.1");
        client.println("Host: bqgbsomhldpfsbgfkgsb.supabase.co");
        client.print("apikey: ");
        client.println(SUPABASE_API_ANON);
        client.println();
        timer = millis();
  }
  if(numberChirp >= 0) {
        if(millis() - chirpTimer < 50) {
        digitalWrite(buzzer, HIGH);
        } else if(millis() - chirpTimer > 100) {
        //Serial.println("decr");
```

```
        chirpTimer = millis();
        numberChirp--;
        } else if(millis() - chirpTimer > 50) {
        digitalWrite(buzzer, LOW);
        }
}

if(sonicStage == 0) {
        digitalWrite(trigPin, LOW);
        sonicStage = 1;
        sonicPinTimer = millis();
} else if(millis() - sonicPinTimer > 2 && sonicStage == 1) {
        digitalWrite(trigPin, HIGH);
        sonicStage = 2;
        sonicPinTimer = millis();
} else if(millis() - sonicPinTimer > 10 && sonicStage == 2) {
        digitalWrite(trigPin, HIGH);
        sonicStage = 0;
        sonicPinTimer = millis();
        // Sets the trigPin on HIGH state for 10 micro seconds

        digitalWrite(trigPin, LOW);
        // Reads the echoPin, returns the sound wave travel time in microseconds
        duration = pulseIn(echoPin, HIGH);
        //  Calculating the distance
        distance = duration * 0.034 / 2;
// Prints the distance on the Serial Monitor
        Serial.println(distance);
        if(distance < 50 && millis()-sonicReportTimer > 60 * 1000) {
        reportEvent("moti");
        sonicReportTimer = millis();
```

```
        }

        int i;
        if(mySerial.available()) {
        char ch=mySerial.read();
        Serial.print(ch);
        if(ch=='{'){
        data[0]=index=0;
        if(gathering == false) {
        gathering = true;
        }
        } else if(ch != '}' && gathering == true) {
        data[index++]=ch; data[index]=0;
        } else if(gathering==true){
        gathering = false;
        data[index++]='\0';
        char* g = data;
        evaluateParams(&g);
        data[0]=index=0;
        }
        }
  }
}
/* just wrap the received data up to 80 columns in the serial print*/
/* -------------------------------------------------------------------------- */
void read_response() {
/* -------------------------------------------------------------------------- */
  uint32_t received_data_num = 0;
  while (client.available()) {
        /* actual data reception */
        char c = client.read();
```

```
        /* print data to serial port */
        Serial.print(c);

  }

}


/* ---------------------------------------------------------------------- */
void printWifiStatus() {
/* ---------------------------------------------------------------------- */
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

ADDITIONAL: SUPABASE EDGE FUNCTION FOR PUSH NOTIFICATION

```
import { createClient } from 'npm:@supabase/supabase-js@2';
import { JWT } from 'npm:google-auth-library@9';
import serviceAccount from '../service-account.json' with {
  type: 'json'
};
```

```
const supabase = createClient(Deno.env.get('SUPABASE_URL'),
Deno.env.get('SUPABASE_SERVICE_ROLE_KEY'));
async function msg(ev) {
  switch(ev){
    case "fail":
      return "Failed attempt to open door detected!";
      break;
    case "touch":
      return "Doorknob touch detected!";
      break;
    case "moti":
      return "Motion detected at door.";
      break;
    default:
      const { data, error } = await supabase.from('test').select('name').eq('id', ev);
      console.log(data[0].name);
      return "Door opened by " + data[0].name;
      break;
  }
}
Deno.serve(async (req)=>{
  const payload = await req.json();
  const { data } = await supabase.from('profiles').select('fcm_token').eq('id',
payload.record.user_id).single();
  const fcmToken = data.fcm_token;
  const accessToken = await getAccessToken({
    clientEmail: serviceAccount.client_email,
    privateKey: serviceAccount.private_key
  });
  const res = await
fetch(`https://fcm.googleapis.com/v1/projects/${serviceAccount.project_id}/messages:se
nd`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${accessToken}`
    },
    body: JSON.stringify({
      message: {
        token: fcmToken,
        notification: {
          title: `ALERT!!!`,
          body: await msg(payload.record.eventID)
        }
      }
    })
  });
```

```
  const resData = await res.json();
  if (res.status < 200 || 299 < res.status) {
    throw resData;
  }
  return new Response(JSON.stringify(resData), {
    headers: {
      'Content-Type': 'application/json'
    }
  });
});
const getAccessToken = ({ clientEmail, privateKey })=>{
  return new Promise((resolve, reject)=>{
    const jwtClient = new JWT({
      email: clientEmail,
      key: privateKey,
      scopes: [
        'https://www.googleapis.com/auth/firebase.messaging'
      ]
    });
    jwtClient.authorize((err, tokens)=>{
      if (err) {
        reject(err);
        return;
      }
      resolve(tokens.access_token);
    });
  });
};
```