

## Overview

In this lesson, you will learn how to apply the RC522 RFID Reader Module on ESP32. This module uses the Serial Peripheral Interface (SPI) bus to communicate with controllers such as Arduino, Raspberry Pi, beagle board, etc.

### Component Required:

- (1) x Elegoo ESP32
- (1) x RC522 RFID module
- (1) x IR receiver module
- (1) x SG90 servo

## Component Introduction

### RC522

**The** MFRC522 is a highly integrated reader/writer for contactless communication at 13.56 MHz. The MFRC522 reader supports ISO 14443A / MIFARE® mode.

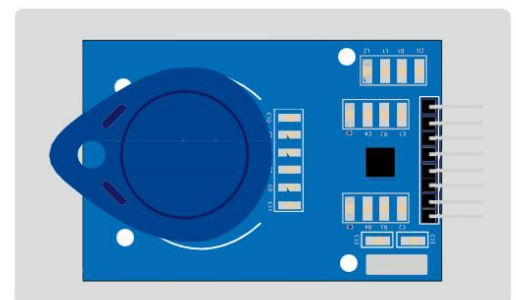
**The** MFRC522's internal transmitter part is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443A/MIFARE® cards and transponders without additional active circuitry. The receiver part provides a robust and efficient implementation of a demodulation

and decoding circuitry for signals from ISO/IEC 14443A/MIFARE® compatible cards and transponders. The digital part handles the complete ISO/IEC 14443A framing and error detection (Parity & CRC). The MFRC522 supports MIFARE®Classic (e.g. MIFARE® Standard) products. The MFRC522 supports contactless communication using MIFARE® higher transfer speeds up to 848 kbit/s in both directions.

**Various** host interfaces are implemented:

- SPI interface
- Serial UART (similar to RS232 with voltage levels according pad voltage supply)
- I2C interface.

**The** figure below shows a typical circuit diagram, using a complementary antenna connection to the MFRC522.



## IR RECEIVER SENSOR:

**IR** detectors are little microchips with a photocell that are tuned to listen to infrared light. They are almost always used for remote control detection - every TV and DVD player has one of these in the front to listen for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels. IR light is not visible to the human eye, which means it takes a little more work to test a setup.

**There** are a few difference between these and say a CdS Photocells:

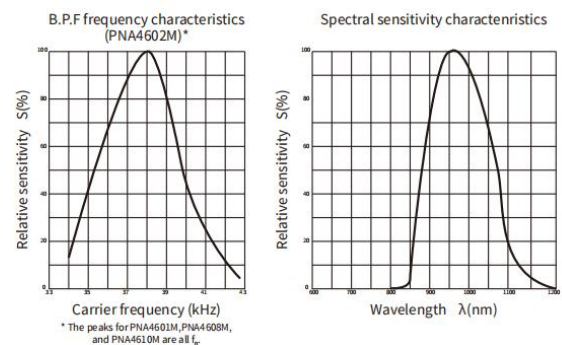
IR detectors are specially filtered for IR light, they are not good at detecting visible light. On the other hand, photocells are good at detecting yellow/green visible light, and are not good at IR light.

**IR** detectors have a demodulator inside that looks for modulated IR at 38 KHz. Just shining an IR LED won't be detected, it has to be PWM blinking at 38KHz. Photocells do not have any sort of demodulator and can detect any frequency (including DC) within the response speed of the photocell (which is about 1KHz)

**IR** detectors are digital out - either they detect 38KHz IR signal and output low (0V) or they do not detect any and output high (5V). Photocells act like resistors, the resistance changes depending on how much light they are exposed to.

**As** you can see from these datasheet graphs, the peak frequency detection is at 38 KHz and the peak LED color is 940 nm. You can use from about 35 KHz to 41 KHz but the sensitivity will drop off so that it won't detect as well from afar. Likewise, you can use 850 to 1100 nm LEDs but they won't work as well as 900 to 1000nm so make sure to get matching LEDs! Check the datasheet for your IR LED to verify the wavelength.

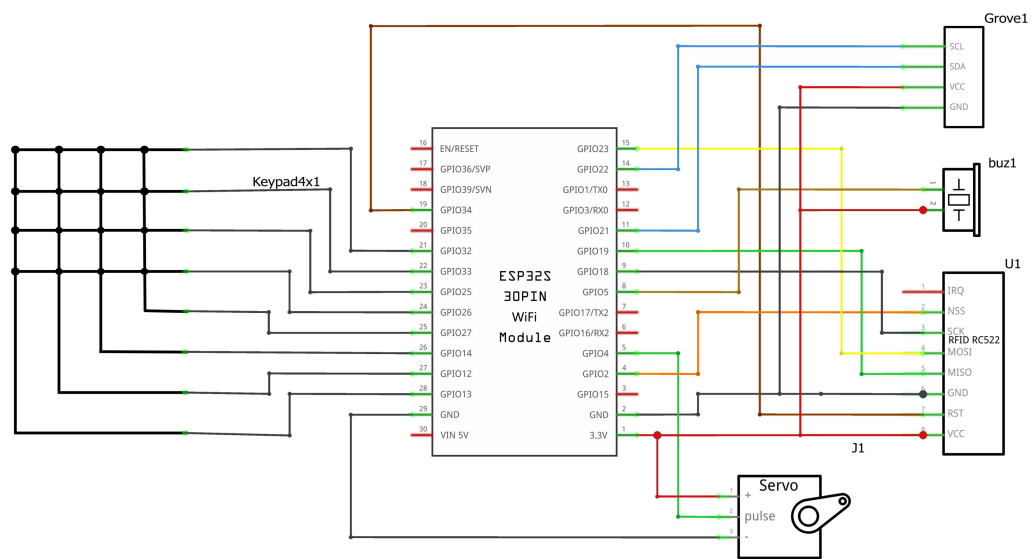
**Try** to get a 940nm - remember that 940nm is not visible light!



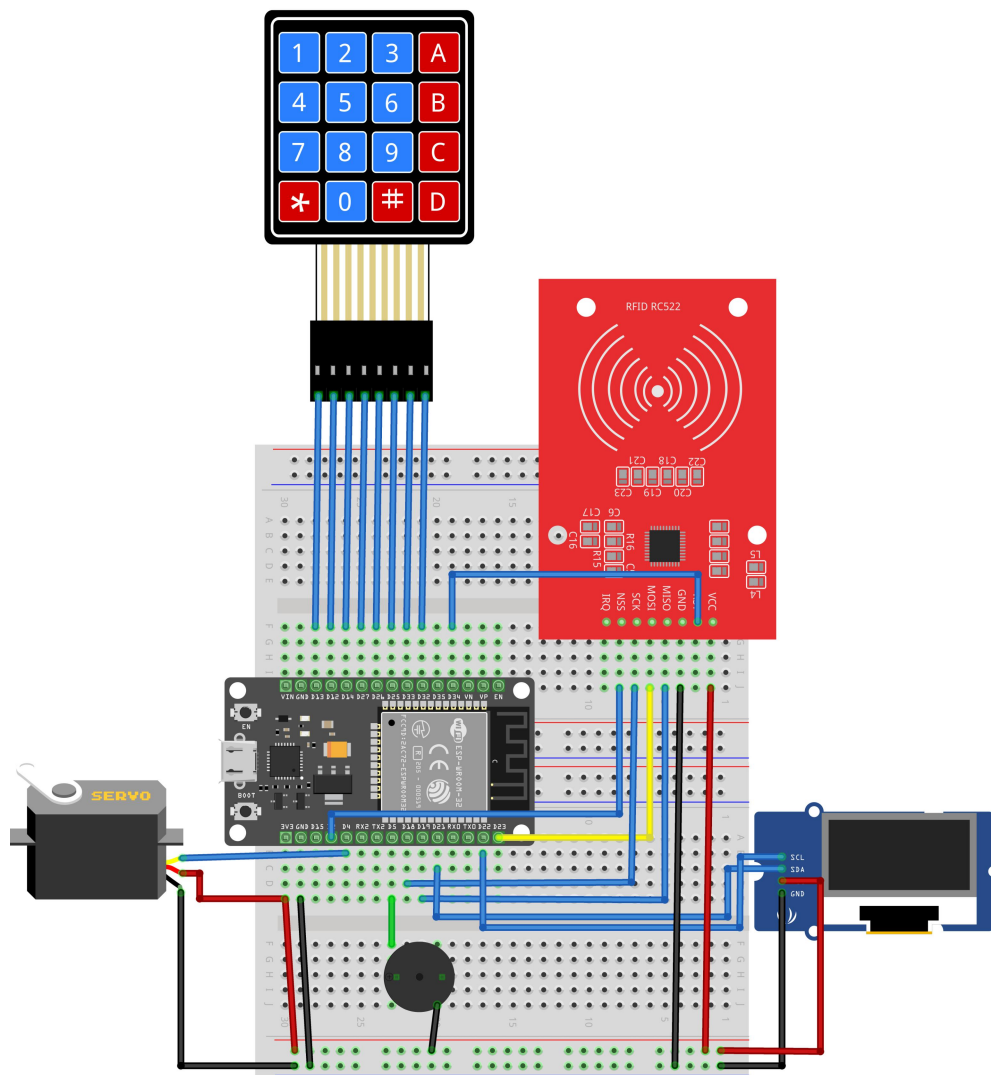
## SG90

- Universal for JR and FP connector
- Cable length : 25cm
- No load; Operating speed: 0.12 sec / 60 degree (4.8V), 0.10 sec / 60 degree (6.0V)
- Stall torque (4.8V): 1.6kg/cm
- Temperature : -30~60'C
- Dead band width: 5us
- Working voltage: 3.5~6V
- Dimension : 1.26 in x 1.18 in x 0.47 in (3.2 cm x 3 cm x 1.2 cm)
- Weight : 4.73 oz (134 g)





## Connection Schematic



Wiring diagram

## Code

After wiring, please open the program in the code folder- **Smart\_Access\_Control** and press UPLOAD to upload the program. See Lesson 5 of part 1 for details about program uploading if there are any errors.

Before you can run this, make sure that you have installed the **< rfid >** and **<servo>** library or re- install it, if necessary. Otherwise, your code won't work.

For details about loading the library file, see Lesson 5 of part 1 .

```
/*
 * Typical pin layout used:
 * -----
 *
 *      MFRC522      Arduino      ESP32      Arduino      Arduino      Arduino
 *      Reader/PCD    Uno          Pin          Nano v3       Leonardo/Micro Pro Micro
 * Signal            Pin          Pin          Pin          Pin          Pin
 * -----
 * RST/Reset         RST           9           34          D9           RESET/ICSP-5   RST
 * SPI SS            SDA(SS)       10          D10         10           10
 * SPI MOSI          MOSI          11 / ICSP-4  23          D11          ICSP-4         16
 * SPI MISO          MISO          12 / ICSP-1  19          D12          ICSP-1         14
 * SPI SCK           SCK           13 / ICSP-3  18          D13          ICSP-3         15
 */
```

**In** the RC522.cpp file, the printCardUID() function is used to print the card ID number. This function outputs the card ID as a four-digit hexadecimal number through the serial port. Please record the four-digit hexadecimal number displayed, as this is the unique ID number of the card, which can be used to determine whether the card has been registered.

```
13:49:05.374 -> Card Detected - UID: 83 E8 8D 04
```

**Within** the RC522 code file, there is an array designated for storing card ID numbers. When a new card ID is read, it must be added to this array in the specified format. It is important to note that the four-digit hexadecimal ID number read must be completed into a full hexadecimal form before being added to the array.

```
const byte authorizedUIDs[][4] = {
  {0x83, 0xE8, 0x8D, 0x04},
  {0x31, 0x1A, 0xCE, 0x05}
};
```

**The** **checkAuthorization** function is used to verify whether the UID of the currently detected IC card is in the predefined “authorization whitelist.” It is the core logic for access control permission judgment. The parameters are described as follows:

- 1.Input parameter **cardUid**: A pointer to the UID of the currently detected card (i.e., mfrc522.uid.uidByte), which is used to pass the UID data to be verified.
- 2.Iterate through the authorization list **authorizedUIDs**: By looping through the list, each authorized UID

in the list is compared one by one with the current card's UID.

3. Efficient comparison logic: The **memcmp()** function is used to compare each byte of the 4 UID bytes (the standard UID length for MIFARE cards is 4 bytes). If the return value of **memcmp()** is 0, it indicates that the current card's UID matches exactly with one of the UIDs in the authorization list, thereby confirming that the card has access control permission.

**In** the **key.cpp**, there are several core functions:

1. Initialize 4x4 matrix keypad and I2C-based OLED display;

```
byte rowPins[ROWS] = {13, 12, 14, 27};  
byte colPins[COLS] = {26, 25, 33, 32};
```

2. The function of 'getkeypad' is to detect the input value from a 4x4 keypad, restrict the input values to digits 0-9, and limit the input length to match the length of the set password, preventing any further input once the limit is exceeded. Support \* key to delete last input digit, # key to trigger password verification;
3. OLED displays input password in real time, shows "Correct/Error" after verification and clears input buffer;
4. Call **unlockDoor()** to unlock when password matches.

**In** the main program **Smart\_Access\_Control.ino**, the key parts to pay attention to are the **handleRFID()**, and **loop()** functions. Specifically:

1. The **handleRFID()** function is used to process the card information read by the RC522 and will activate the servo to open the gate after successful verification.
2. Handle IR priority and RC522 checks in the main loop: Call **getkeypad()** to process IR signals and Check RC522 every 100ms, limiting SPI communication frequency to avoid IR interference