

Forum:[Hack Forums](#)

Thread: [Magnetic Door Lock](#)

Spookz - September 11, 2024, 6:56 am

I've developed a basic prototype of an electromagnetic door lock using a Raspberry Pi. The lock works well, but I've noticed that the electromagnetic component can become warm to the touch after a few days of continuous power. To prevent overheating, I'm considering adding a secondary locking mechanism for long-term use. I'm still evaluating the pros and cons of different options to determine the best solution. Implementing FaceID will take more time due to challenges related to varying lighting conditions and overall complexity. I'll share updates on that as progress is made.

PARTS LIST

[Raspberry PI 4 Model B 4GB](#)

[GPIO Extension Board](#)

[400 Point Bread Board](#)

[Electric Magnetic Lock](#)

[12V 1A Power Supply](#)

[4x4 Membrane Keypad](#)

[Jumper Wires](#)

[5V Low-Level Trigger Relay](#)

[I2C LCD Module](#)

[Electromagnetic Buzzer](#)

```
Code import I2C_LCD_driver
import RPi.GPIO as GPIO
from time import sleep
```

```
# Pin configuration
COLUMNS = [5, 6, 13, 19]
ROWS = [12, 16, 20, 21]
BUZZER = 17
RELAY = 27
SECRET_CODE = "1234"
```

```
# Initialize variables
relay_state = True
keypad_pressed = -1
input_code = ""
```

```
# Setup GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.setup(RELAY, GPIO.OUT)
GPIO.output(RELAY, GPIO.HIGH)
```

```
for pin in COLUMNS:
```

```

GPIO.setup(pin, GPIO.OUT)
for pin in ROWS:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# Initialize LCD
lcd = I2C_LCD_driver.lcd()
lcd.lcd_display_string("System loading", 1, 1)
for a in range(16):
    lcd.lcd_display_string(".", 2, a)
    sleep(0.1)
lcd.lcd_clear()

# Callback for keypad press
def keypad_callback(channel):
    global keypad_pressed
    if keypad_pressed == -1:
        keypad_pressed = channel

# Register event detection for rows
for row in ROWS:
    GPIO.add_event_detect(row, GPIO.RISING, callback=keypad_callback)

# Set all columns to a specific state
def set_all_columns(state):
    for col in COLUMNS:
        GPIO.output(col, state)

# Handle commands for checking or clearing PIN
def handle_commands():
    global relay_state, input_code
    pressed = False
    GPIO.output(COLUMNS[0], GPIO.HIGH)

    if GPIO.input(ROWS[0]) == 1:
        lcd.lcd_clear()
        lcd.lcd_display_string("Clear", 1, 5)
        sleep(1)
        pressed = True
    elif GPIO.input(ROWS[1]) == 1:
        if input_code == SECRET_CODE:
            lcd.lcd_clear()
            lcd.lcd_display_string("Successful", 1, 3)
            relay_state = not relay_state
            GPIO.output(RELAY, GPIO.LOW if relay_state else GPIO.HIGH)
            GPIO.output(BUZZER, GPIO.HIGH)
            sleep(0.3)
            GPIO.output(BUZZER, GPIO.LOW)
            sleep(1)
        else:
            lcd.lcd_clear()
            lcd.lcd_display_string("Wrong PIN!", 1, 3)
            for _ in range(3):
                GPIO.output(BUZZER, GPIO.HIGH)

```

```

        sleep(0.3)
        GPIO.output(BUZZER, GPIO.LOW)
        sleep(0.3)
    pressed = True

    GPIO.output(COLUMNS[0], GPIO.LOW)
    if pressed:
        input_code = ""
    return pressed

# Read input from keypad
def read_input(column, characters):
    global input_code
    GPIO.output(column, GPIO.HIGH)
    for i, row in enumerate(ROWS):
        if GPIO.input(row) == 1:
            input_code += characters[i]
            lcd lcd_display_string(input_code, 2, 0)
    GPIO.output(column, GPIO.LOW)

# Main loop
try:
    while True:
        lcd lcd_display_string("Enter your PIN:", 1, 0)
        if keypad_pressed != -1:
            set_all_columns(GPIO.HIGH)
            if GPIO.input(keypad_pressed) == 0:
                keypad_pressed = -1
            else:
                sleep(0.1)
        else:
            if not handle_commands():
                read_input(COLUMNS[0], ["D", "C", "B", "A"])
                read_input(COLUMNS[1], ["#", "9", "6", "3"])
                read_input(COLUMNS[2], ["0", "8", "5", "2"])
                read_input(COLUMNS[3], ["*", "7", "4", "1"])
                sleep(0.1)
            else:
                sleep(0.1)
except KeyboardInterrupt:
    print("Stopped!")

```

Pentester™ - October 17, 2024, 10:03 am

I like the idea of creating your own device but for about \$40 you can buy one and put it on a basic 12/24v power supply depending on what you purchase and use a wireless relay device that can fit on a keychain, Won't have any issues with overheating

Azeta - November 26, 2024, 4:36 am

Crafty stuff. Like the idea.

Pentester™ - January 4, 2025, 11:14 am

(Sep 11, 2024 06:56 AM)Spookz Wrote: [\[CITE\]](#) I've developed a basic prototype of an electromagnetic door lock using a Raspberry Pi. The lock works well, but I've noticed that the

electromagnetic component can become warm to the touch after a few days of continuous power. To prevent overheating, I'm considering adding a secondary locking mechanism for long-term use. I'm still evaluating the pros and cons of different options to determine the best solution. Implementing FaceID will take more time due to challenges related to varying lighting conditions and overall complexity. I'll share updates on that as progress is made.

PARTS LIST

[Raspberry PI 4 Model B 4GB](#)

[GPIO Extension Board](#)

[400 Point Bread Board](#)

[Electric Magnetic Lock](#)

[12V 1A Power Supply](#)

[4x4 Membrane Keypad](#)

[Jumper Wires](#)

[5V Low-Level Trigger Relay](#)

[I2C LCD Module](#)

[Electromagnetic Buzzer](#)

```
Code import I2C_LCD_driver
import RPi.GPIO as GPIO
from time import sleep
```

```
# Pin configuration
COLUMNS = [5, 6, 13, 19]
ROWS = [12, 16, 20, 21]
BUZZER = 17
RELAY = 27
SECRET_CODE = "1234"
```

```
# Initialize variables
relay_state = True
keypad_pressed = -1
input_code = ""
```

```
# Setup GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.setup(RELAY, GPIO.OUT)
GPIO.output(RELAY, GPIO.HIGH)
```

```
for pin in COLUMNS:
    GPIO.setup(pin, GPIO.OUT)
for pin in ROWS:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

```
# Initialize LCD
lcd = I2C_LCD_driver.lcd()
lcd.lcd_display_string("System loading", 1, 1)
for a in range(16):
    lcd.lcd_display_string(".", 2, a)
```

```

    sleep(0.1)
    lcd.lcd_clear()

# Callback for keypad press
def keypad_callback(channel):
    global keypad_pressed
    if keypad_pressed == -1:
        keypad_pressed = channel

# Register event detection for rows
for row in ROWS:
    GPIO.add_event_detect(row, GPIO.RISING, callback=keypad_callback)

# Set all columns to a specific state
def set_all_columns(state):
    for col in COLUMNS:
        GPIO.output(col, state)

# Handle commands for checking or clearing PIN
def handle_commands():
    global relay_state, input_code
    pressed = False
    GPIO.output(COLUMNS[0], GPIO.HIGH)

    if GPIO.input(ROWS[0]) == 1:
        lcd.lcd_clear()
        lcd.lcd_display_string("Clear", 1, 5)
        sleep(1)
        pressed = True
    elif GPIO.input(ROWS[1]) == 1:
        if input_code == SECRET_CODE:
            lcd.lcd_clear()
            lcd.lcd_display_string("Successful", 1, 3)
            relay_state = not relay_state
            GPIO.output(RELAY, GPIO.LOW if relay_state else GPIO.HIGH)
            GPIO.output(BUZZER, GPIO.HIGH)
            sleep(0.3)
            GPIO.output(BUZZER, GPIO.LOW)
            sleep(1)
        else:
            lcd.lcd_clear()
            lcd.lcd_display_string("Wrong PIN!", 1, 3)
            for _ in range(3):
                GPIO.output(BUZZER, GPIO.HIGH)
                sleep(0.3)
                GPIO.output(BUZZER, GPIO.LOW)
                sleep(0.3)
            pressed = True

    GPIO.output(COLUMNS[0], GPIO.LOW)
    if pressed:
        input_code = ""
    return pressed

```

```

# Read input from keypad
def read_input(column, characters):
    global input_code
    GPIO.output(column, GPIO.HIGH)
    for i, row in enumerate(ROWS):
        if GPIO.input(row) == 1:
            input_code += characters[i]
            lcd lcd_display_string(input_code, 2, 0)
    GPIO.output(column, GPIO.LOW)

# Main loop
try:
    while True:
        lcd lcd_display_string("Enter your PIN:", 1, 0)
        if keypad_pressed != -1:
            set_all_columns(GPIO.HIGH)
            if GPIO.input(keypad_pressed) == 0:
                keypad_pressed = -1
            else:
                sleep(0.1)
        else:
            if not handle_commands():
                read_input(COLUMNS[0], ["D", "C", "B", "A"])
                read_input(COLUMNS[1], ["#", "9", "6", "3"])
                read_input(COLUMNS[2], ["0", "8", "5", "2"])
                read_input(COLUMNS[3], ["*", "7", "4", "1"])
                sleep(0.1)
            else:
                sleep(0.1)
except KeyboardInterrupt:
    print("Stopped!")

```

Also I forgot to mention instead of using a maglock you can use a door strike lock, They have them fail safe and fail secure depending on your usage and they're made for any door style

ttbr0 - January 4, 2025, 4:47 pm

if anyone cares, i thought i would just mention that theres probably a timing vulnerability in your implementation. the way you compare the input code string using the == operator uses a memcmp underneath (check the cpython source code) which compares the string byte by byte (probably must implementation on your pi). the problem with this is that if it fails on a specific byte, the function will return early and you can measure the time it takes when this happens. this is a side channel timing attack that can be brute forced.

proper string comparison checking should be constant time like this:

```

Code def secure_strncmp(s1: str, s2: str, length: int):
    result = True

    for i in range(length):
        result &= s1[i] == s2[i]

    return result

```

sid 7asan - January 6, 2025, 9:13 am

(Sep 11, 2024 06:56 AM)Spookz Wrote: [\[CITE\]](#) I've developed a basic prototype of an electromagnetic door lock using a Raspberry Pi. The lock works well, but I've noticed that the electromagnetic component can become warm to the touch after a few days of continuous power. To prevent overheating, I'm considering adding a secondary locking mechanism for long-term use. I'm still evaluating the pros and cons of different options to determine the best solution. Implementing FaceID will take more time due to challenges related to varying lighting conditions and overall complexity. I'll share updates on that as progress is made.

PARTS LIST

[Raspberry PI 4 Model B 4GB](#)

[GPIO Extension Board](#)

[400 Point Bread Board](#)

[Electric Magnetic Lock](#)

[12V 1A Power Supply](#)

[4x4 Membrane Keypad](#)

[Jumper Wires](#)

[5V Low-Level Trigger Relay](#)

[I2C LCD Module](#)

[Electromagnetic Buzzer](#)

```
Code import I2C_LCD_driver
import RPi.GPIO as GPIO
from time import sleep
```

```
# Pin configuration
COLUMNS = [5, 6, 13, 19]
ROWS = [12, 16, 20, 21]
BUZZER = 17
RELAY = 27
SECRET_CODE = "1234"
```

```
# Initialize variables
relay_state = True
keypad_pressed = -1
input_code = ""
```

```
# Setup GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.setup(RELAY, GPIO.OUT)
GPIO.output(RELAY, GPIO.HIGH)
```

```
for pin in COLUMNS:
    GPIO.setup(pin, GPIO.OUT)
for pin in ROWS:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

```
# Initialize LCD
lcd = I2C_LCD_driver.lcd()
```

```

lcd.lcd_display_string("System loading", 1, 1)
for a in range(16):
    lcd.lcd_display_string(".", 2, a)
    sleep(0.1)
lcd.lcd_clear()

# Callback for keypad press
def keypad_callback(channel):
    global keypad_pressed
    if keypad_pressed == -1:
        keypad_pressed = channel

# Register event detection for rows
for row in ROWS:
    GPIO.add_event_detect(row, GPIO.RISING, callback=keypad_callback)

# Set all columns to a specific state
def set_all_columns(state):
    for col in COLUMNS:
        GPIO.output(col, state)

# Handle commands for checking or clearing PIN
def handle_commands():
    global relay_state, input_code
    pressed = False
    GPIO.output(COLUMNS[0], GPIO.HIGH)

    if GPIO.input(ROWS[0]) == 1:
        lcd.lcd_clear()
        lcd.lcd_display_string("Clear", 1, 5)
        sleep(1)
        pressed = True
    elif GPIO.input(ROWS[1]) == 1:
        if input_code == SECRET_CODE:
            lcd.lcd_clear()
            lcd.lcd_display_string("Successful", 1, 3)
            relay_state = not relay_state
            GPIO.output(RELAY, GPIO.LOW if relay_state else GPIO.HIGH)
            GPIO.output(BUZZER, GPIO.HIGH)
            sleep(0.3)
            GPIO.output(BUZZER, GPIO.LOW)
            sleep(1)
        else:
            lcd.lcd_clear()
            lcd.lcd_display_string("Wrong PIN!", 1, 3)
            for _ in range(3):
                GPIO.output(BUZZER, GPIO.HIGH)
                sleep(0.3)
                GPIO.output(BUZZER, GPIO.LOW)
                sleep(0.3)
            pressed = True

    GPIO.output(COLUMNS[0], GPIO.LOW)

```

```

if pressed:
    input_code = ""
    return pressed

# Read input from keypad
def read_input(column, characters):
    global input_code
    GPIO.output(column, GPIO.HIGH)
    for i, row in enumerate(ROWS):
        if GPIO.input(row) == 1:
            input_code += characters[i]
            lcd lcd_display_string(input_code, 2, 0)
    GPIO.output(column, GPIO.LOW)

# Main loop
try:
    while True:
        lcd lcd_display_string("Enter your PIN:", 1, 0)
        if keypad_pressed != -1:
            set_all_columns(GPIO.HIGH)
            if GPIO.input(keypad_pressed) == 0:
                keypad_pressed = -1
            else:
                sleep(0.1)
        else:
            if not handle_commands():
                read_input(COLUMNS[0], ["D", "C", "B", "A"])
                read_input(COLUMNS[1], ["#", "9", "6", "3"])
                read_input(COLUMNS[2], ["0", "8", "5", "2"])
                read_input(COLUMNS[3], ["*", "7", "4", "1"])
                sleep(0.1)
            else:
                sleep(0.1)
except KeyboardInterrupt:
    print("Stopped!")
great job bro :) ??

```