EED397: IOT(Internet of Things)

Intruder Capture System

Professor: Dr Rohit Singh

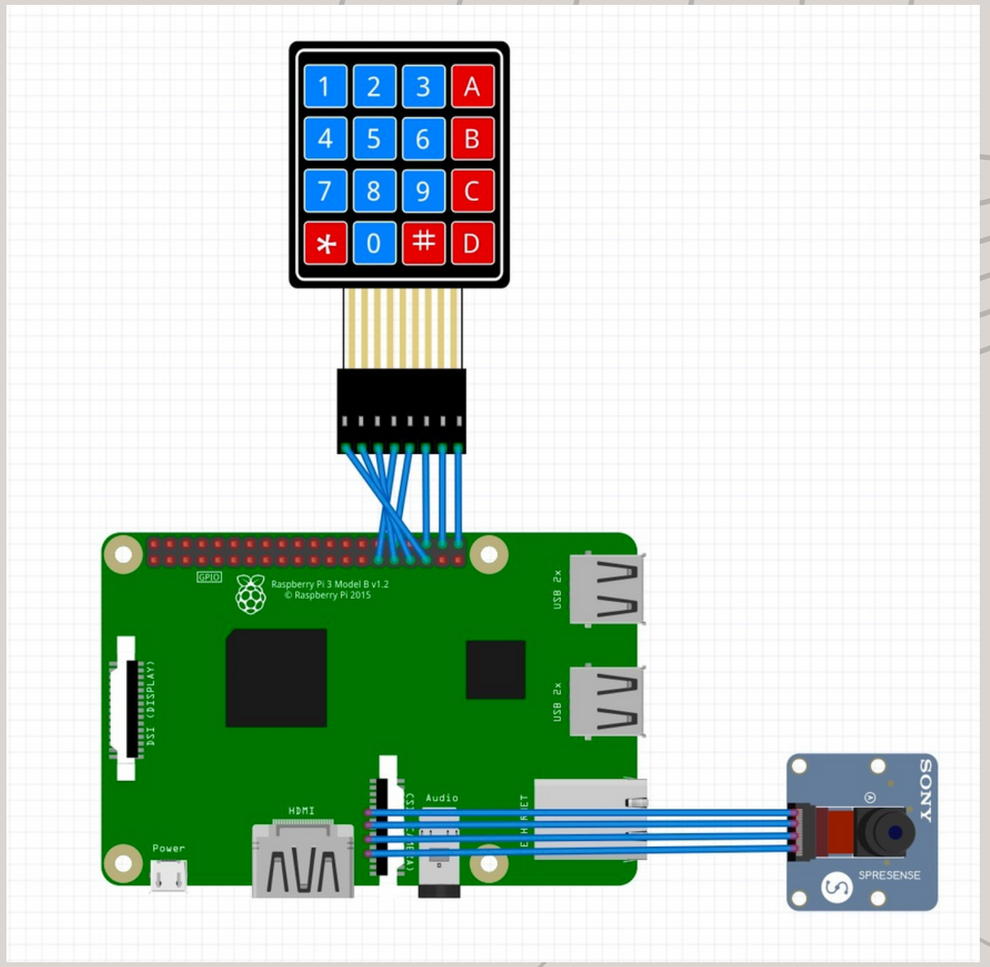
Akshay Veeragandham
Suchit Reddi
Kolli Sreeram Sai
Koteswara Bezawada

Abstract

- This project introduces an IoT-based Intruder Capture System utilizing the MQTT protocol for efficient communication. Powered by a Raspberry Pi, it manages access and intrusion detection.
- In the event of an incorrect password attempt, a camera captures the intruder's photo, sent in real-time to a subscribing device for verification.
- Integrating hardware and software, the Raspberry Pi handles MQTT messaging, resulting in a scalable, robust door security system.
- This system enhances safety and smart access control, showcasing the synergy of IoT in real-world applications.

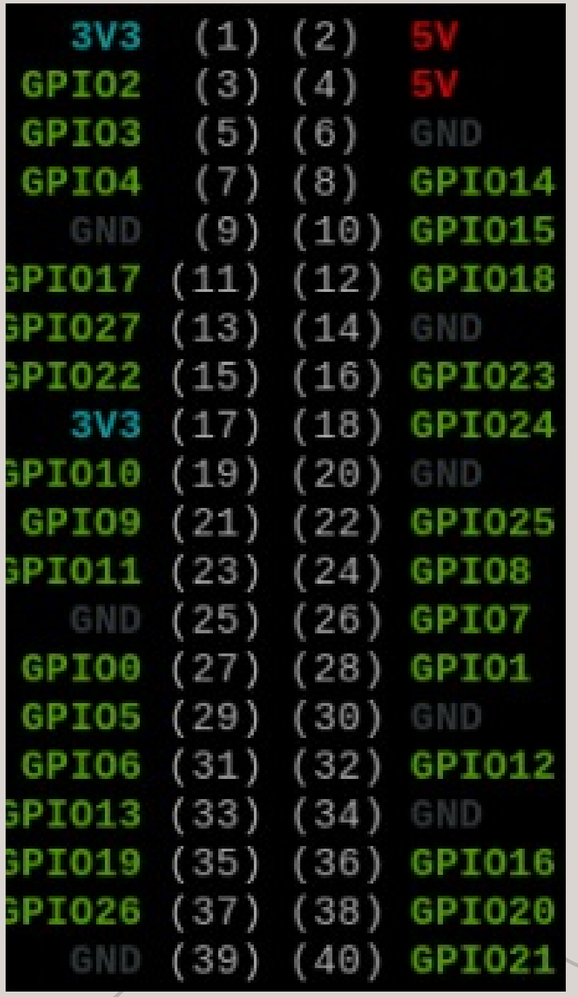
Components

- Raspberry pi
- Camera Module
- Numpad
- MQTT Protocol



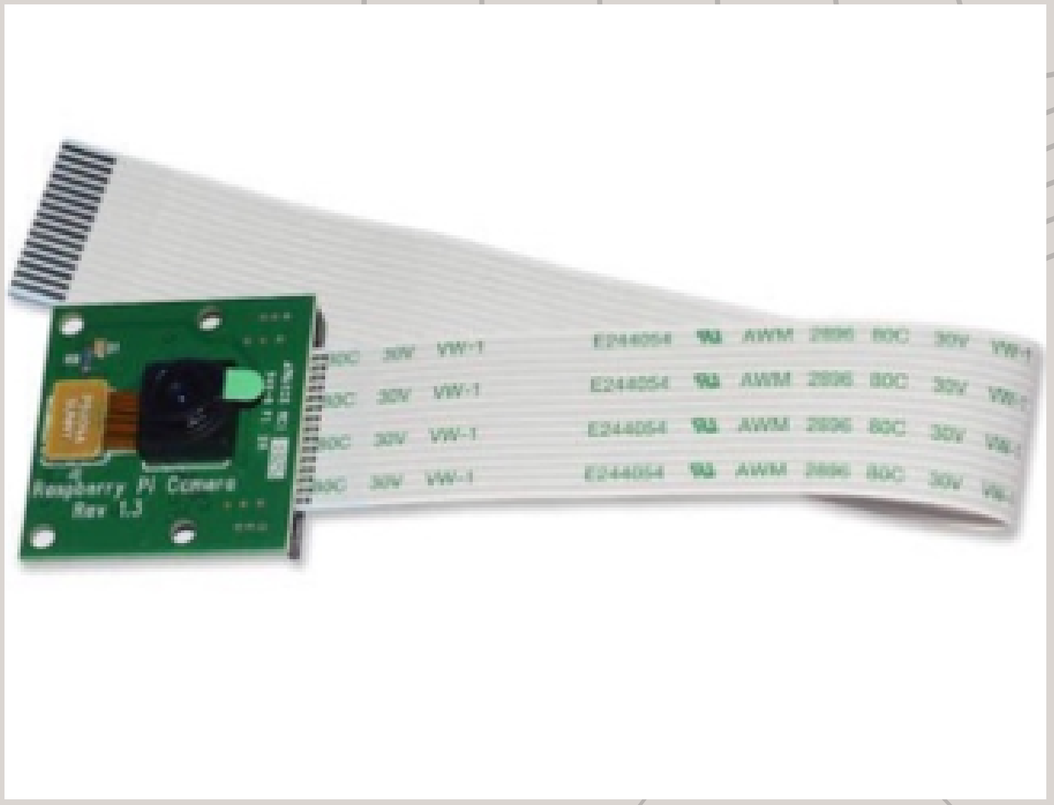
Circuit Connections

- We are using 8 GPIO pins for interfacing the matrix keypad module.
- The GPIO pins 5,6,13,19 are used for lines and 12,16,20,21 for columns.
- We are using the GPIO.BCM pin mode.
- A camera module is connected to the CSI camera connector.



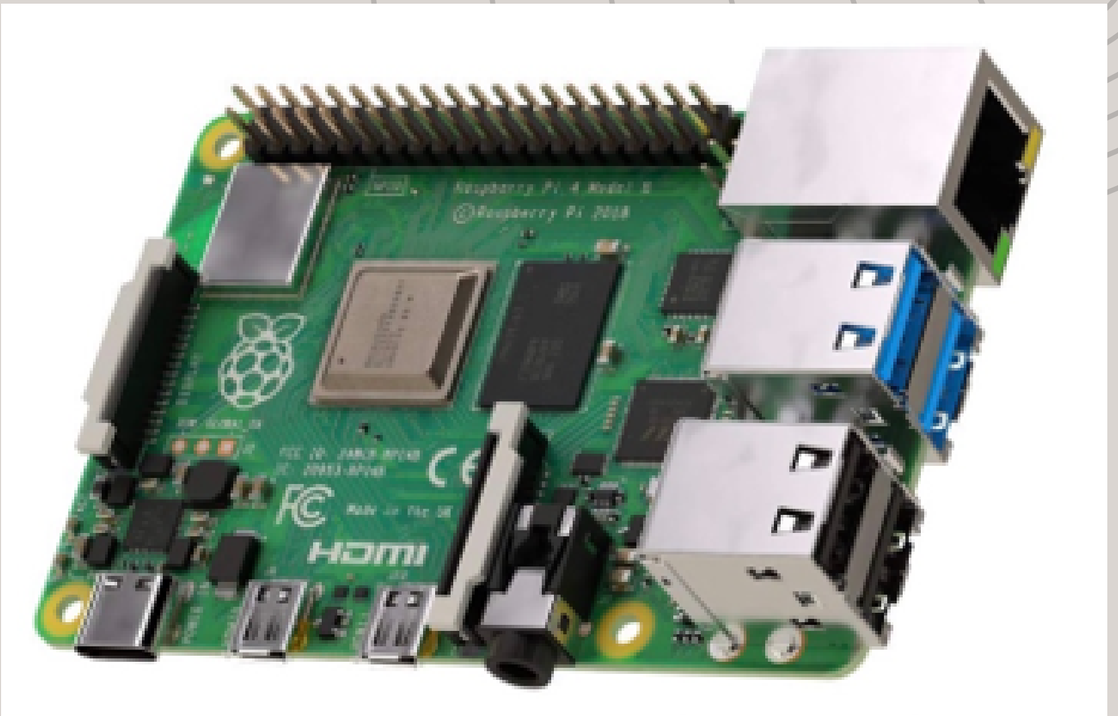
Camera Module

Enhancing the security measures of the Smart Door Security System, the camera module is activated upon incorrect password attempts. Capturing images in real-time, the camera module visually verifies individuals seeking access. This visual data not only adds an extra layer of authentication but also facilitates real-time monitoring and alerts in the event of unauthorized access.



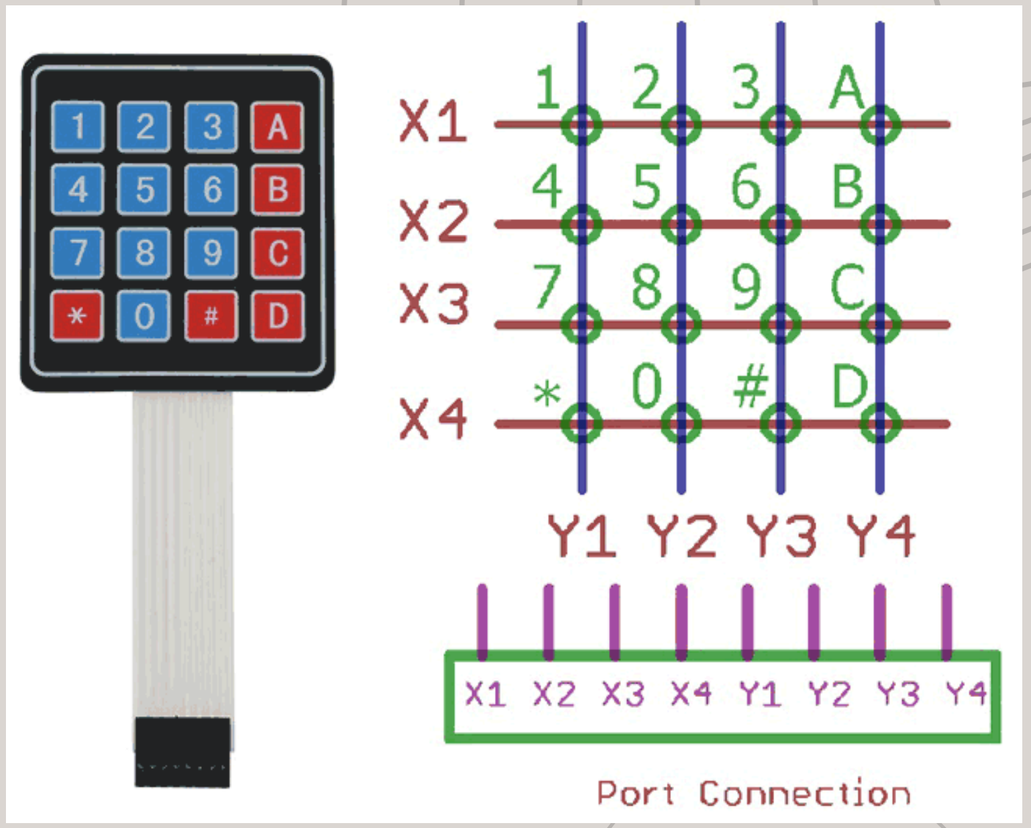
Raspberry Pi

At the heart of the Smart Door Security System lies the Raspberry Pi, functioning as the central control unit. This versatile single-board computer is responsible for managing access control, processing keypad input, and activating the camera module. The GPIO pins of the Raspberry Pi are utilized to interface with the system's peripherals, creating a seamless integration between hardware and software.



4x4 Matrix Keypad

The keypad serves as the primary user interface for entering access codes into the system. Interfacing with the Raspberry Pi, the keypad provides a secure and user-friendly means of input. Its integration is crucial for enabling convenient access control, allowing users to input their access codes with ease.



Matrix Keypad Code

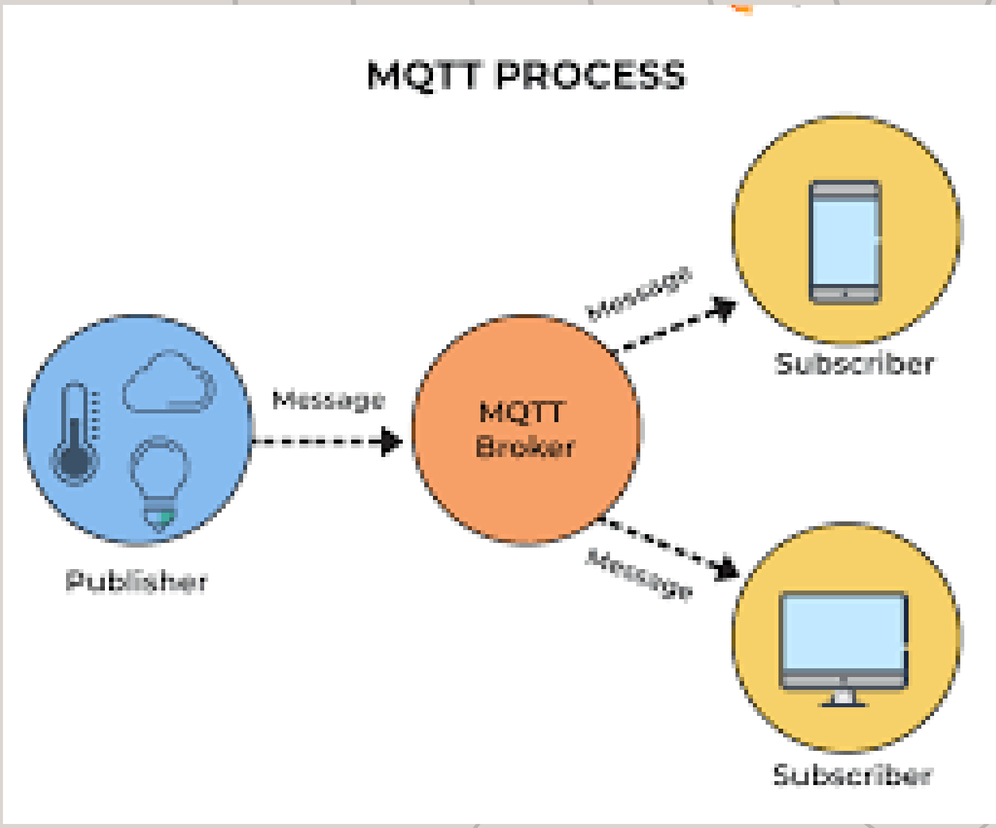
```
# The GPIO pin of the column of the key held down or -1 if no key is pressed
keypadPressed = -1

print("Please enter your code")
secretCode = "7897"
input = ""

try:
    while True:
        # If a button was pressed, check whether it is released
        if keypadPressed != -1:
            setAllLines(GPIO.HIGH)
            if GPIO.input(keypadPressed) == 0:
                keypadPressed = -1
            else:
                time.sleep(0.1)
        # Otherwise, just read the input
        else:
             if len(input) != len(secretCode):
                readLine(L1, ["1","2","3","A"])
                readLine(L2, ["4","5","6","B"])
                readLine(L3, ["7","8","9","C"])
                readLine(L4, ["*","0","#","D"])
                time.sleep(0.1)
             else:
                 if input == secretCode :
                     print("Correct");
                     input=""
                 else:
                     print("Incorrect!!");
                     input="";
                     os.system("mkdir img")
                     os.system("libcamera-jpeg -o ./img/intruder_captured.jpeg --vflip=1");
                     os.system("convert -resize 10% ./img/intruder_captured.jpeg ./img/intruder_compressed.jpeg");
                     print("Redirecting to send.py"); os.system("venv/bin/python3 send.py")
                #Anything after this doesn't get executed
```

MQTT Protocol

The MQTT (Message Queuing Telemetry Transport) protocol acts as the communication backbone of the system. This lightweight and efficient protocol enable secure and real-time data exchange between devices. In the context of the Smart Door Security System, Mosquitto Mqtt is utilized for transmitting images captured by the camera module. This ensures that subscribing devices receive prompt alerts and visual verification in response to security events.



Working Code

Sender Code

```python
import paho.mqtt.client as mqtt
import os

topic = "suchit/image"

def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")

    f=open("./img/intruder_compressed.jpeg", "rb")
    fileContent = f.read()
    byteArr = bytearray(fileContent)
    #print(byteArr)
    client.publish(topic, byteArr, 2)
    print("Published")
    print("Redirecting to numpad.py")
    os.system("venv/bin/python3 numpad.py")
    #Anything after this line does not execute.

client = mqtt.Client()
client.on_connect = on_connect
client.connect("test.mosquitto.org", 1883, 60)

client.loop_forever()
```

Reciever Code

```python
import paho.mqtt.client as mqtt
import os

topic = "suchit/image"

def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    # Subscribe, which need to put into on_connect
    # If reconnect after losing the connection with the broker,
    # it will continue to subscribe to the suchit/image topic
    client.subscribe(topic)
    print("Subscribed to " + topic)

# The callback function, it will be triggered when receiving messages
def on_message(client, userdata, msg):
    #print(f"{msg.topic} {msg.payload}")
    os.system("mkdir img")
    f = open('./img/intruder_received.jpeg', "wb")
    f.write(msg.payload)
    print("Image Received")
    f.close()

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# When network experiences a disgraceful disconnect, send last will
client.will_set('suchit/image', b'{"status": "Off"}')
client.connect("test.mosquitto.org", 1883, 60)
client.loop_forever()
```

Challenges

Dashboard & Receiving:

- Most of the time, the MQTT message payloads are text, either a small block of text or a JSON payload of data. That said, it is possible for devices to send files in the MQTT message as a big block of binary data.
- The issue is that most MQTT clients have issues receiving MQTT messages that aren't text, and freak out when the payloads are binary or files.
- **Solution**: Python script that subscribes to the MQTT messages, and then saves the contents of those messages as binary files.

Image Constraints & remote passcode managing

Image size:-

We were not able to send the original image that was captured by the Pi camera as the image was around 1 MB. The subscriber is not able to read and create an image file as the size is considerably large.
Solution: We used a compression software or algorithm.

Password changing:-

Users not being able to set the access code for the matrix keypad is not user-friendly. The user should be able to remotely set the access code.

Conclusion

- In conclusion, the Intruder Capture System presents a commendable solution for secure access control within the realm of the IoT.
- While facing challenges such as image size constraints and limited password change accessibility, the system excels in providing real-time alerts and maintaining a user-friendly interaction.
- Future iterations could focus on image compression refinement, code optimization, biometric integration, improved password management, and integration with broader smart home ecosystems.

References

- GitHub: (*https://github.com/SuchitReddi/ics*)
- Paho MQTT Library Documentation:
- Eclipse Paho - MQTT for Python. (*https://www.eclipse.org/paho/clients/python/docs/*)
- Libcamera Documentation:
- Libcamera - A camera stack for embedded systems. (*https://libcamera.org/docs/*)
- ImageMagick Documentation: (*https://imagemagick.org/script/index.php*)
- Blog Post: "Files Over MQTT - Solution in Python.” (*https://davidmac.pro/posts/2021-07-21-files-over-mqtt/*)


THANK
YOU