



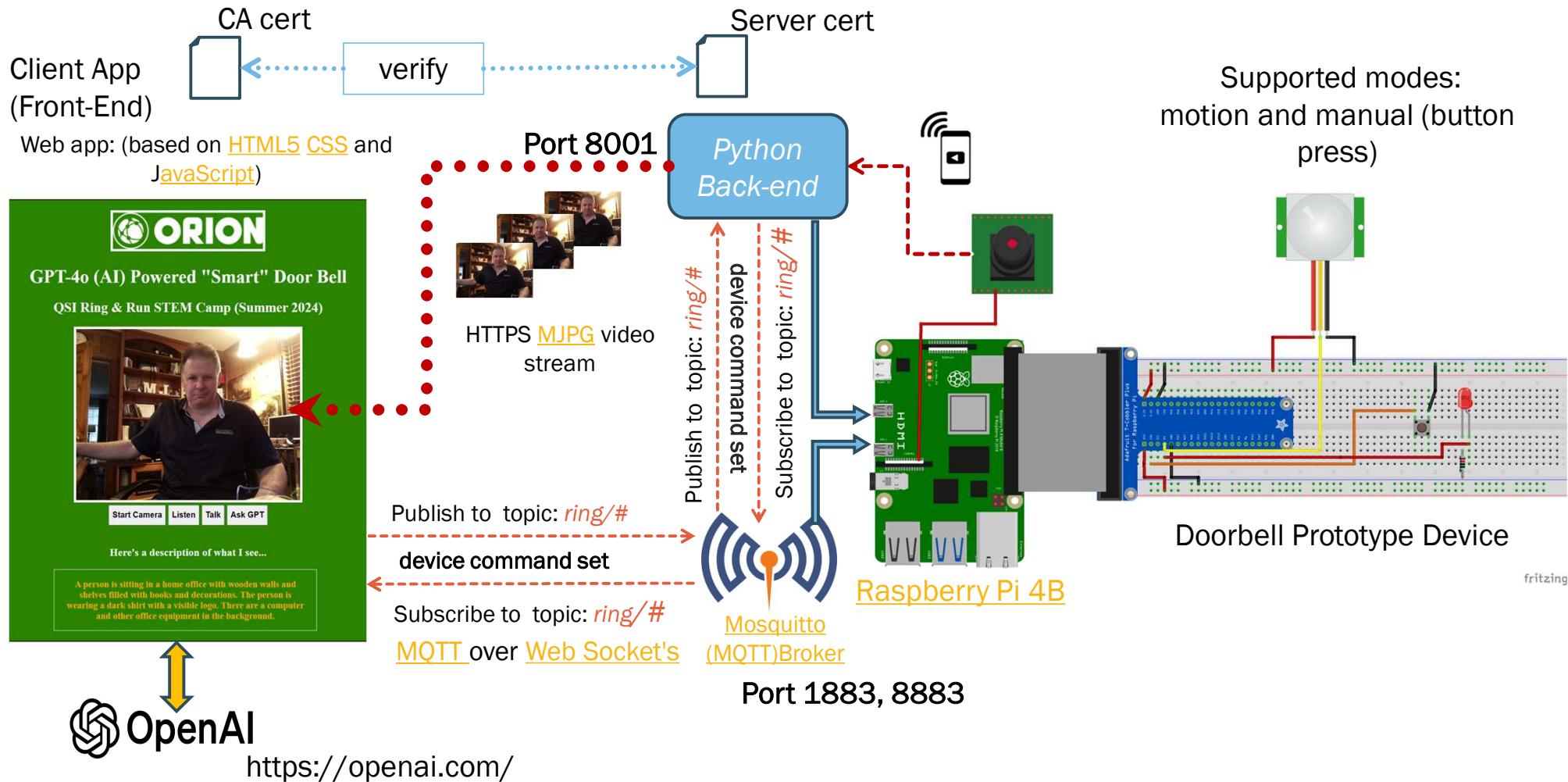
MODULE 3: IMPLEMENT THE “SMART” DOORBELL SOFTWARE

RING & RUN STEM EVENT.



ORION
OPEN ARCHITECTURE RESILIENT IOT
FOR OPERATIONAL NETWORKS

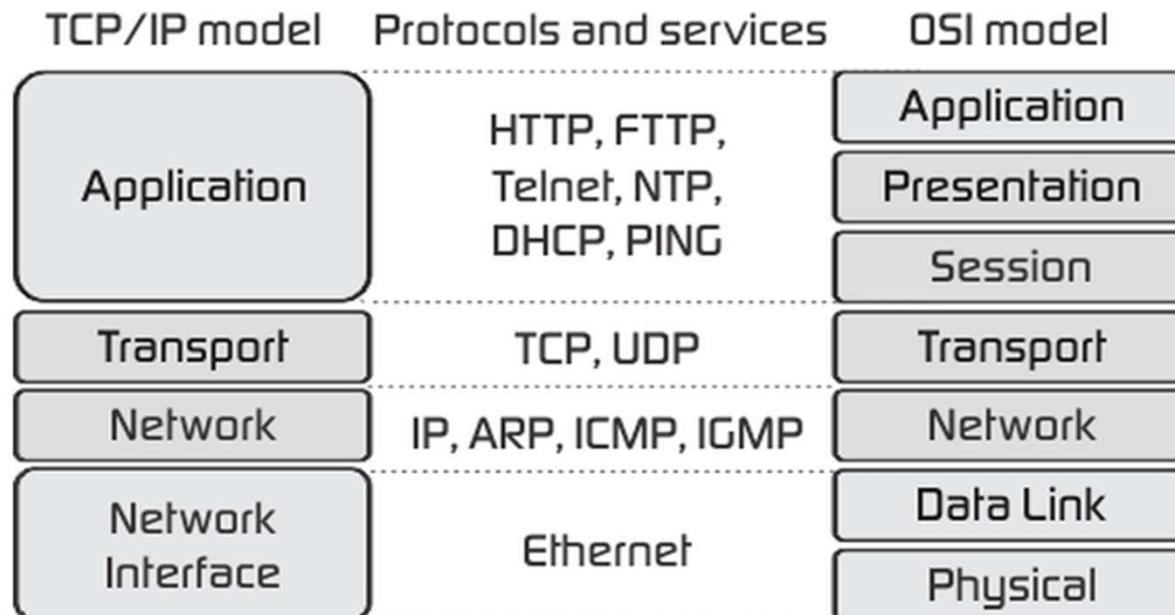
IOT ENABLED “SMART” DOORBELL CONCEPT: SETTING DEVICE CONTROL WITH MQTT



Interlude: Network Protocol Primer

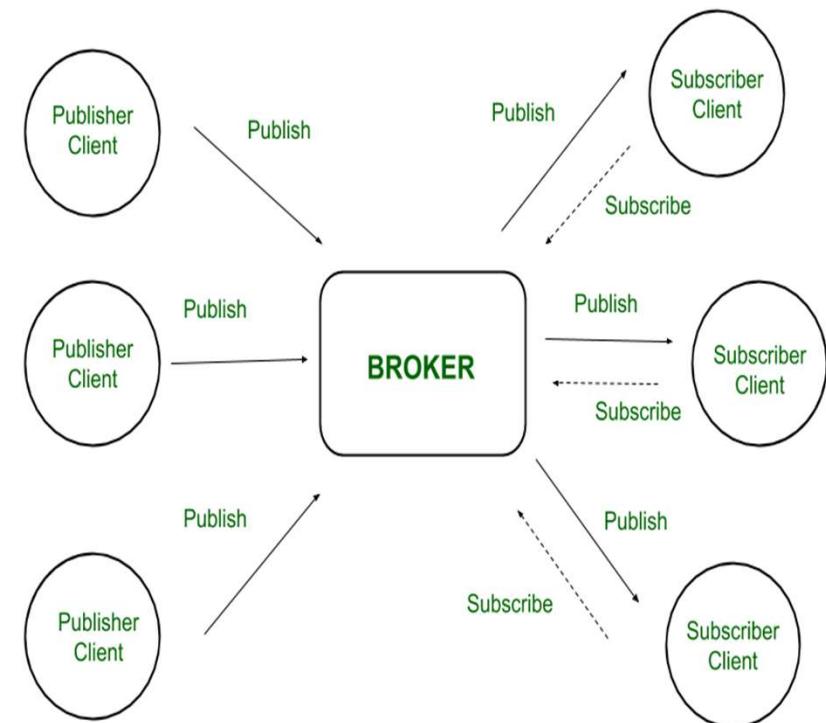
TCP/IP, HTTP, HTTPS (TLS), MQTT

- Computer Networks: <https://www.youtube.com/watch?v=kn7ei2ENJbl>
- Video: (TCP/IP): <https://www.youtube.com/watch?v=vKFLgmSC6do>
- Video: https://www.youtube.com/watch?v=PpsEaqJV_A0
- Video: <https://www.youtube.com/watch?v=P6SZLcGE4us>



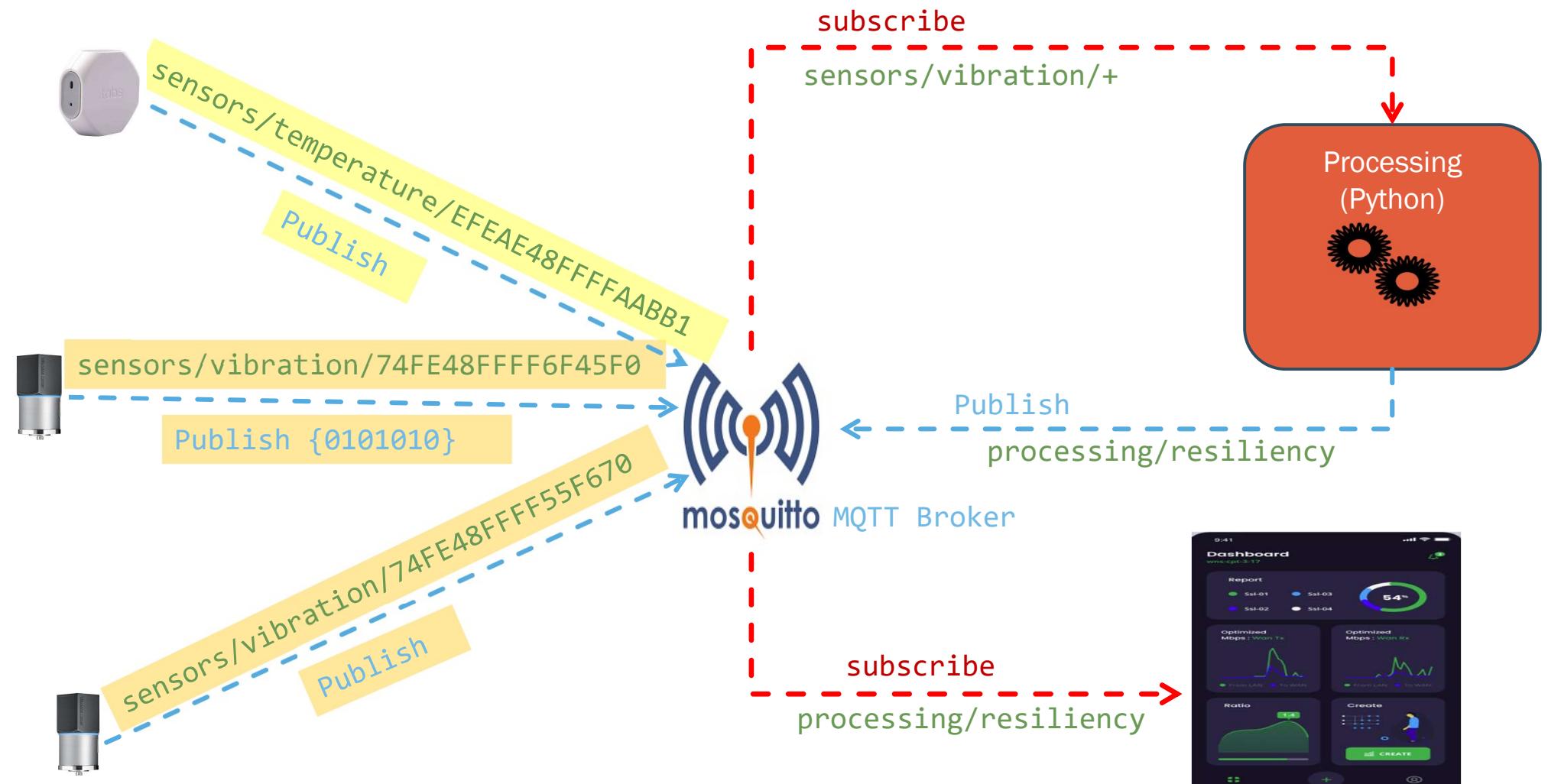
INTEGRATE MQTT SUPPORT

- MQTT stands for Message Queuing Telemetry Transport
 - it's a simple messaging protocol, designed for constrained devices with low bandwidth. - [1]
 - Perfect for exchanging data between IoT devices.
 - MQTT video:
<https://www.youtube.com/watch?v=eS4nx6tLSLs>
 - <https://mqtt.org/>
 - <https://learn.sparkfun.com/tutorials/introduction-to-mqtt/all>
 - <https://www.geeksforgeeks.org/fundamental-features-of-mqtt/>
 - <https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>
 - MQTT over web sockets (JavaScript) tutorial
 - <http://www.steves-internet-guide.com/using-javascript-mqtt-client-websockets/>



Source: <https://www.geeksforgeeks.org/introduction-of-message-queue-telemetry-transport-protocol-mqtt/> - [1]

MQTT Concept Overview



MQTT Publish/Subscribe Topic Structure

```
match individual (single) sensor subscriptions
# sensors/vibration/74FE48FFFF7A1C6F
# sensors/vibration/74FE48FFFF6F4AD1
# sensors/vibration/74FE48FFFF6F45F0
# sensors/vibration/74FE48FFFF55F670

# sensors/sound/E8E1E1000105034E
# sensors/sound/E8E1E1000105036D

# match all levels of hierarchy after # (all sensors)
# sensors/#

# match a single level of hierarchy after + (all vibration sensors)
# sensors/vibration/+

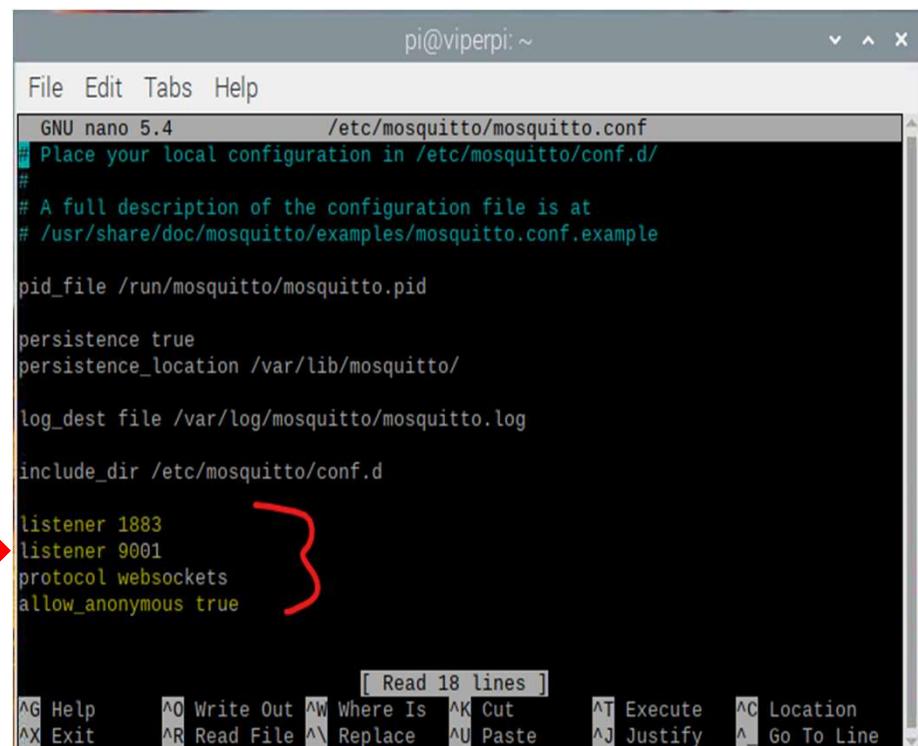
# match a single level of hierarchy after + (all sound sensors)
# sensors/sound/+
```

MQTT BROKER CONFIGURATION

- Configure the MQTT Broker to use Web sockets
 - Start (or use existing) SSH session with the Raspberry Pi
 - Install Mosquitto MQTT Broker

```
sudo apt install -y mosquitto mosquitto-clients
```
 - Configure Mosquitto auto start when the Raspberry Pi boots
 - `sudo systemctl enable mosquitto.service`
 - Use *nano* or *vi* to open the configuration file:

```
sudo nano /etc/mosquitto/mosquitto.conf
```
- Append the highlighted configuration to the end of the and press save:
listener 1883
listener 9001
protocol websockets
allow_anonymous true
A red arrow points from the configuration text to the screenshot of the nano editor.
- For save operation with *nano*, you need to press `Ctrl+o` (lowercase “o”) `<enter>` (which is not intuitive).
- Restart the service: `sudo systemctl restart mosquitto`
 - Note: if the command returns a message, it indicates a problem with configuration file



```
pi@viperpi: ~
File Edit Tabs Help
GNU nano 5.4 /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
listener 9001
protocol websockets
allow_anonymous true }
```

[Read 18 lines]

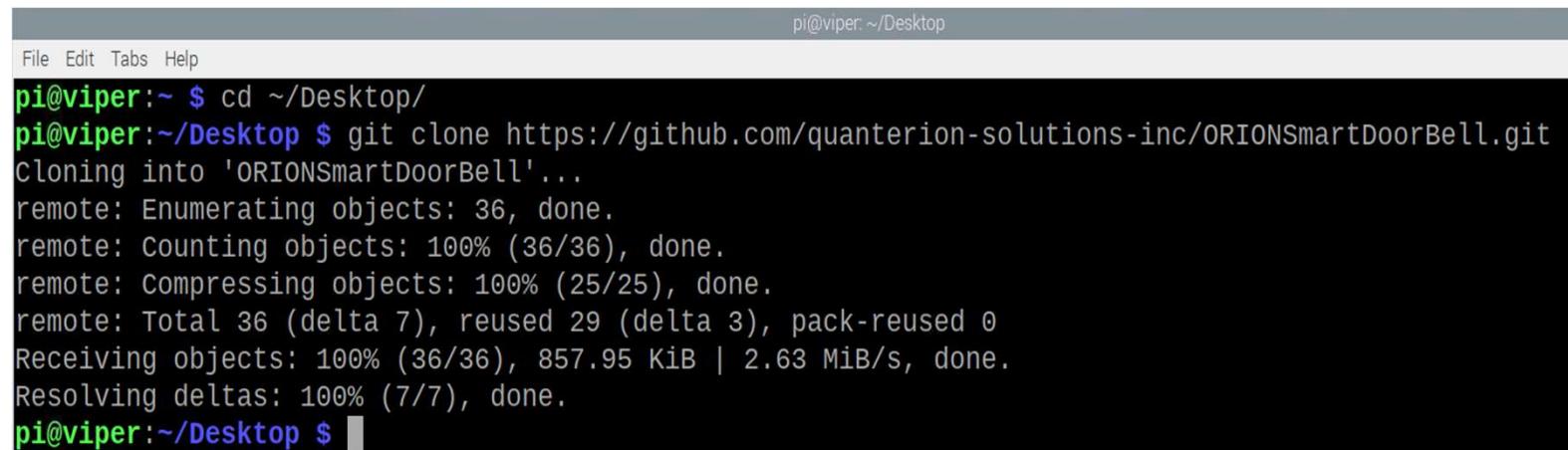
^G Help ^O Write Out ^W Where Is ^K Cut
^X Exit ^R Read File ^X Replace ^U Paste ^T Execute ^C Location
^J Justify ^L Go To Line

Download the Doorbell Project Source Code

1. Establish a new VNC or SSH to the Raspberry Pi
2. Open a command terminal and navigate to the Desktop
 - clone the GitHub project repository with the following command:

```
cd ~/Desktop
```

```
git clone https://github.com/quanterion-solutions-inc/ORIONSmartDoorBell.git
```

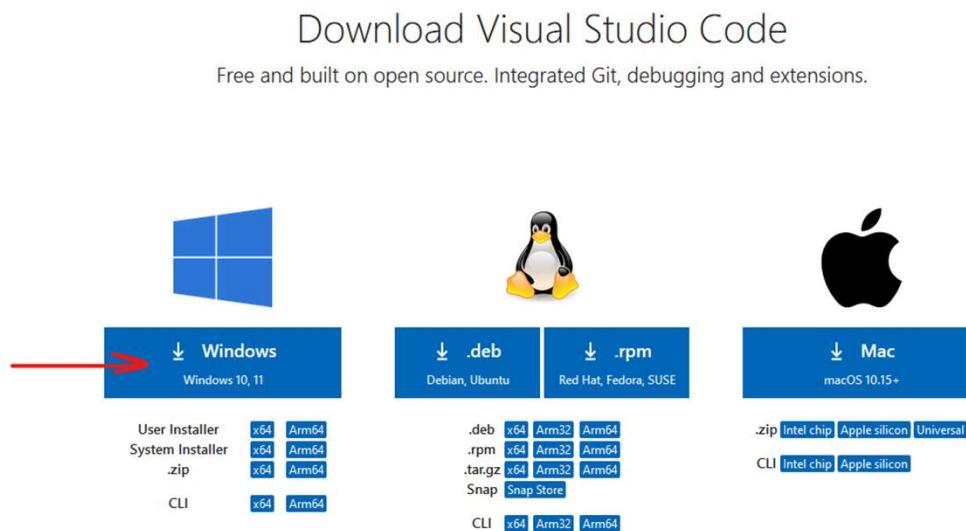


A screenshot of a terminal window titled "pi@viper:~/Desktop". The window shows a command-line interface with the following text:

```
File Edit Tabs Help  
pi@viper:~ $ cd ~/Desktop/  
pi@viper:~/Desktop $ git clone https://github.com/quanterion-solutions-inc/ORIONSmartDoorBell.git  
Cloning into 'ORIONSmartDoorBell'...  
remote: Enumerating objects: 36, done.  
remote: Counting objects: 100% (36/36), done.  
remote: Compressing objects: 100% (25/25), done.  
remote: Total 36 (delta 7), reused 29 (delta 3), pack-reused 0  
Receiving objects: 100% (36/36), 857.95 KiB | 2.63 MiB/S, done.  
Resolving deltas: 100% (7/7), done.  
pi@viper:~/Desktop $ █
```

Install Visual Studio Code (VSCode) Editor

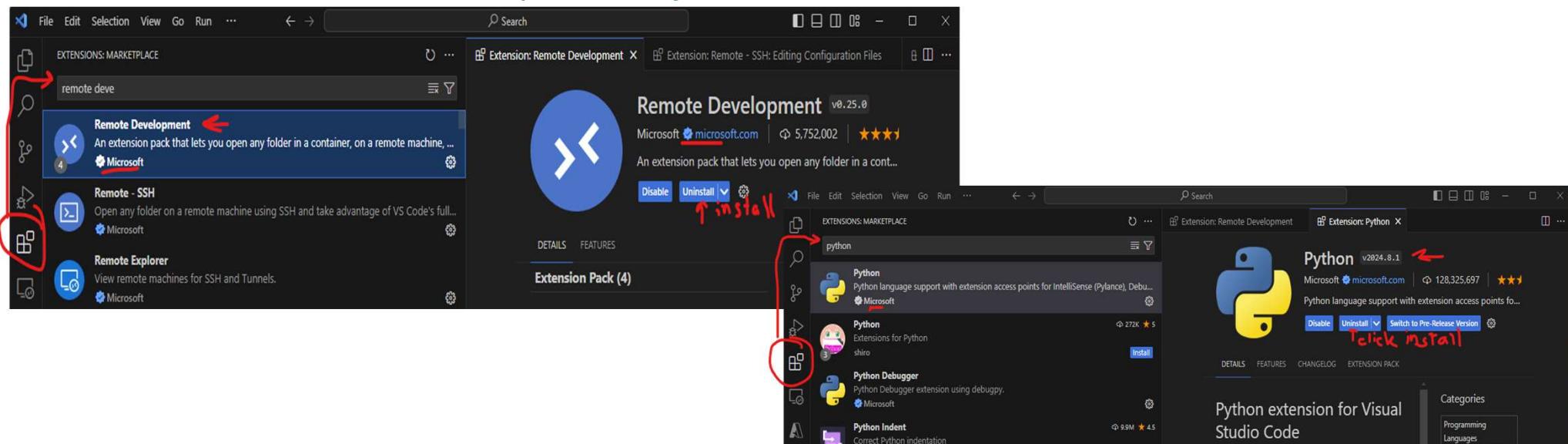
- Download and install Visual studio code for Windows to your development laptop:
 - <https://code.visualstudio.com/download>
 - VSCode is the most popular software development code editor in use today.



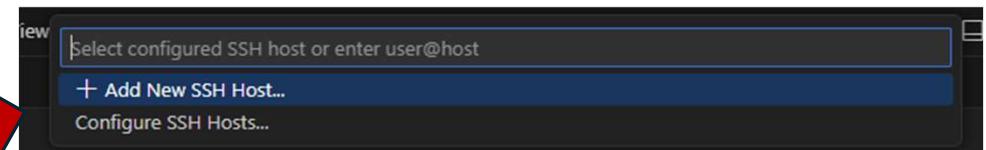
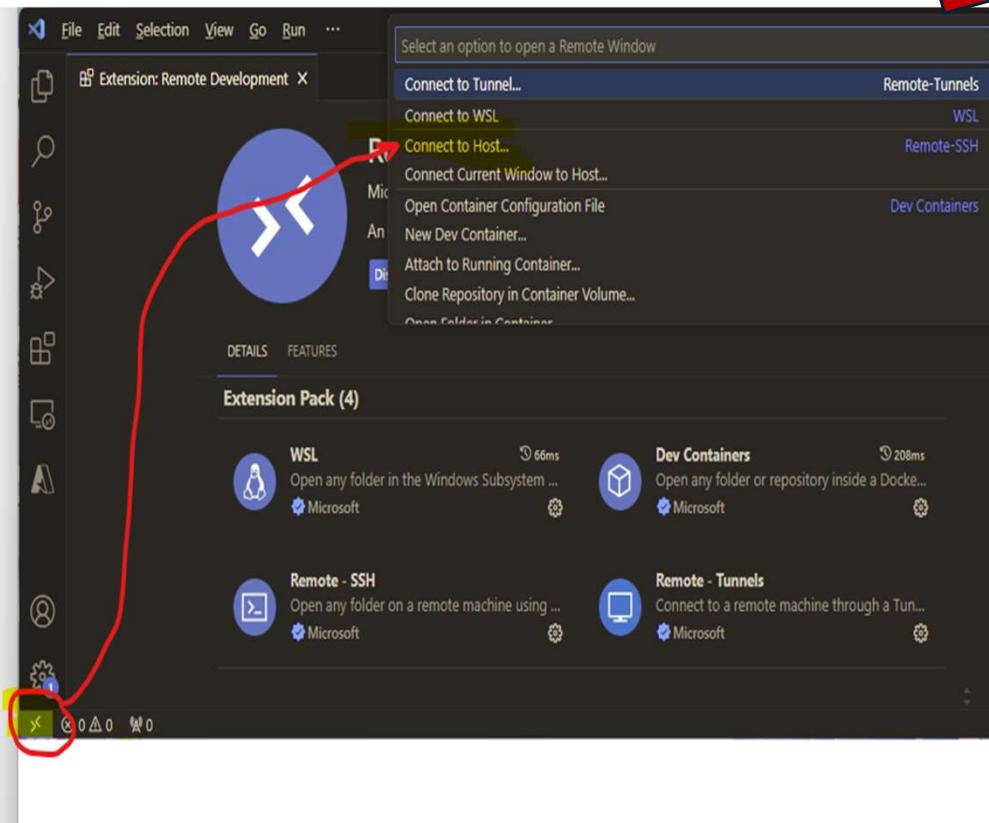
Install VSCode Extensions

- Click the Extensions and type the extension name in search box to install the following extensions (see illustration)
 1. “Remote Development extension pack” -- provides an SSH managed session for transparent remote develop
 2. “Python” - provides support for IntelliSense (Pylance), debugging (Python Debugger), formatting, linting, code navigation, refactoring, variable explorer, test explorer, and more!

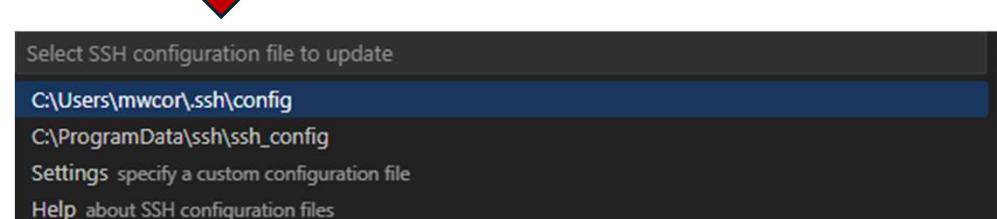
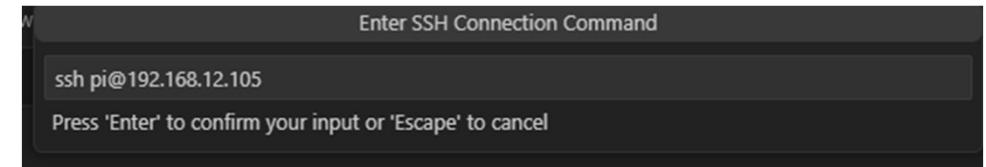
Note: there are many extensions with similar names provided by 3rd parties. Be certain the extensions installed are those provided by Microsoft



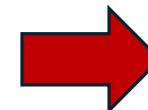
Establish a Remote Connection



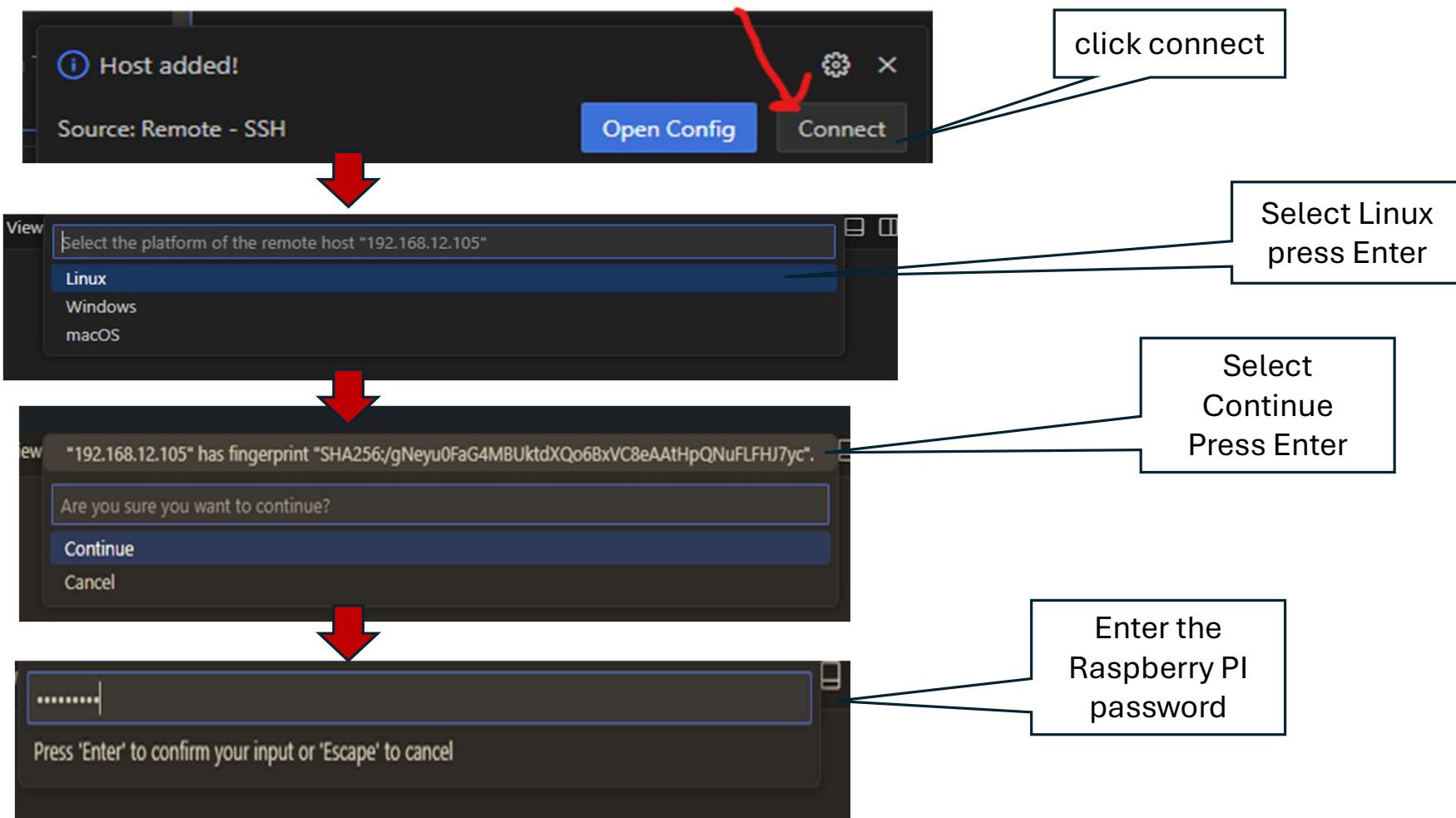
Enter familiar “ssh” command:
use the IP address of Raspberry Pi (as shown)



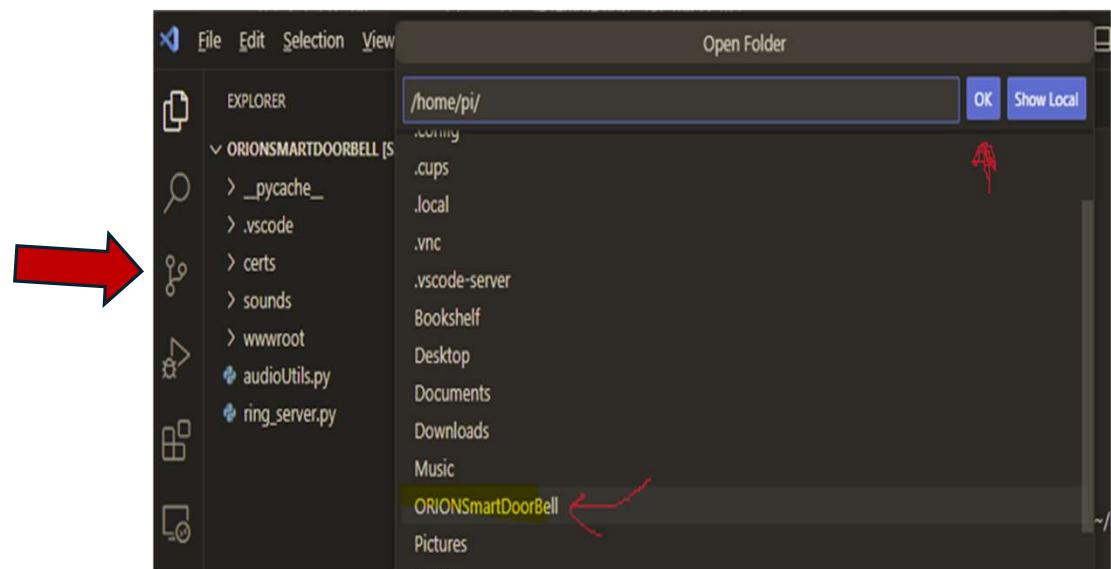
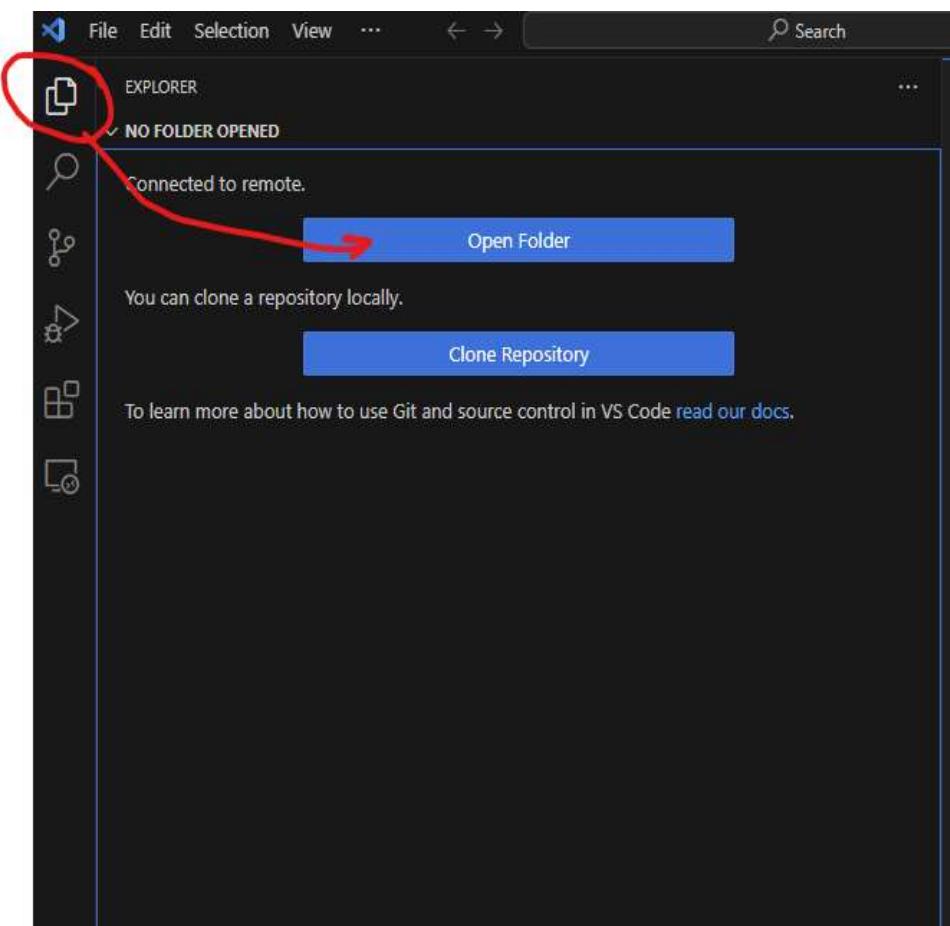
Next slide



- Establish Remote Connection (cont.)



Open the Doorbell Git Project (source code) Folder

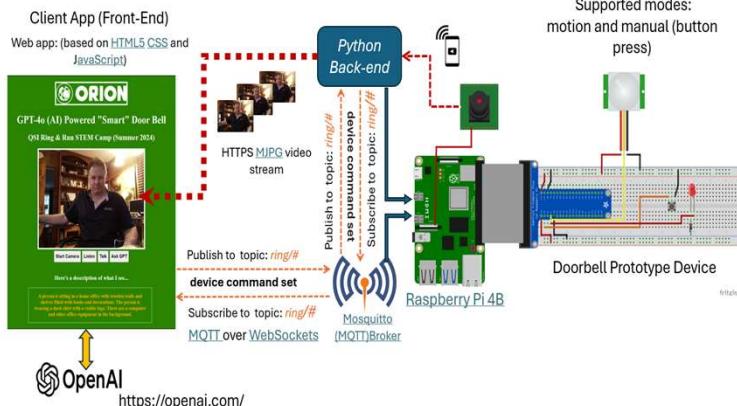


Remote Development Host

* Note: this view being served from the Raspberry Pi to your Remote Dev Host Laptop over a managed SSH session

Doorbell code consists of two parts:

1. Client Application (front-end)
2. Server (back-end)



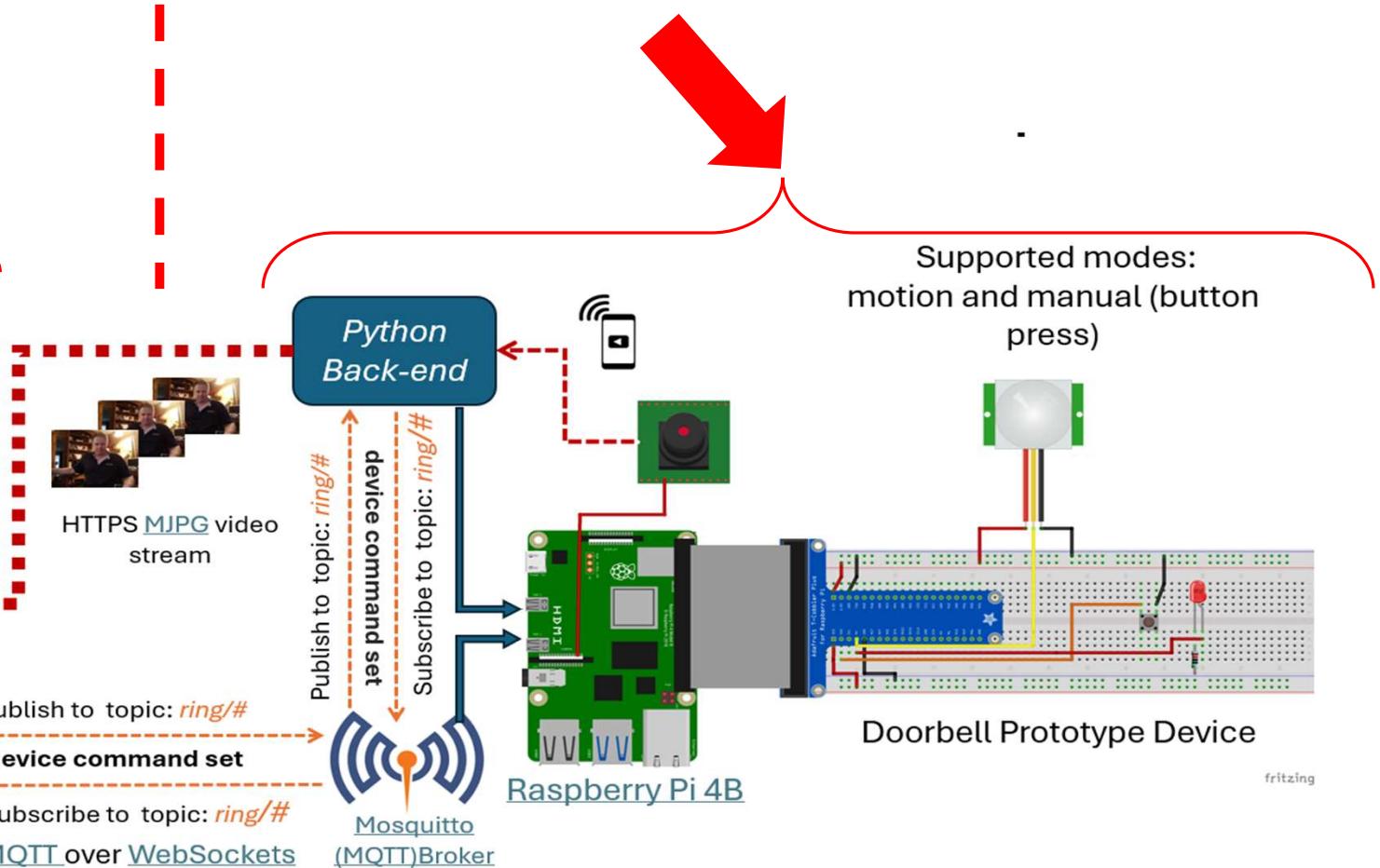
A screenshot of a code editor (VS Code) showing the `ring_server.py` file. The file contains Python code for a streaming output class. The code imports various modules and defines a class `StreamingOutput` with methods for initialization, writing to the buffer, and handling frame notifications.

```
File Edit Selection View Go Run Terminal Help
EXPLORER ORIONSMARTDOORBELL [SSH: 192.168.12.105]
... Welcome ring_server.py
ring_server.py
1 import io
2 import json
3 import picamera
4 from gpiozero import Button, MotionSensor
5 import logging
6 import socketserver
7 from threading import Condition
8 from http import server
9
10 import paho.mqtt.client as paho
11 import base64
12 import requests
13 import time
14
15 import pygame
16 import argparse
17
18 import threading
19
20
21 from audioUtils import *
22 import ssl
23 import subprocess
24
25
26 class StreamingOutput(object):
27     def __init__(self):
28         self.frame = None
29         self.buffer = io.BytesIO()
30         self.condition = Condition()
31
32     def write(self, buf):
33         if buf.startswith(b'\xff\xd8'):
34             #new frame, copy the existing buffer's content and notify all
35             #clients it's available
36             self.buffer.truncate()
37             with self.condition:
38                 self.condition.notify_all()
```

Front-end (web) App

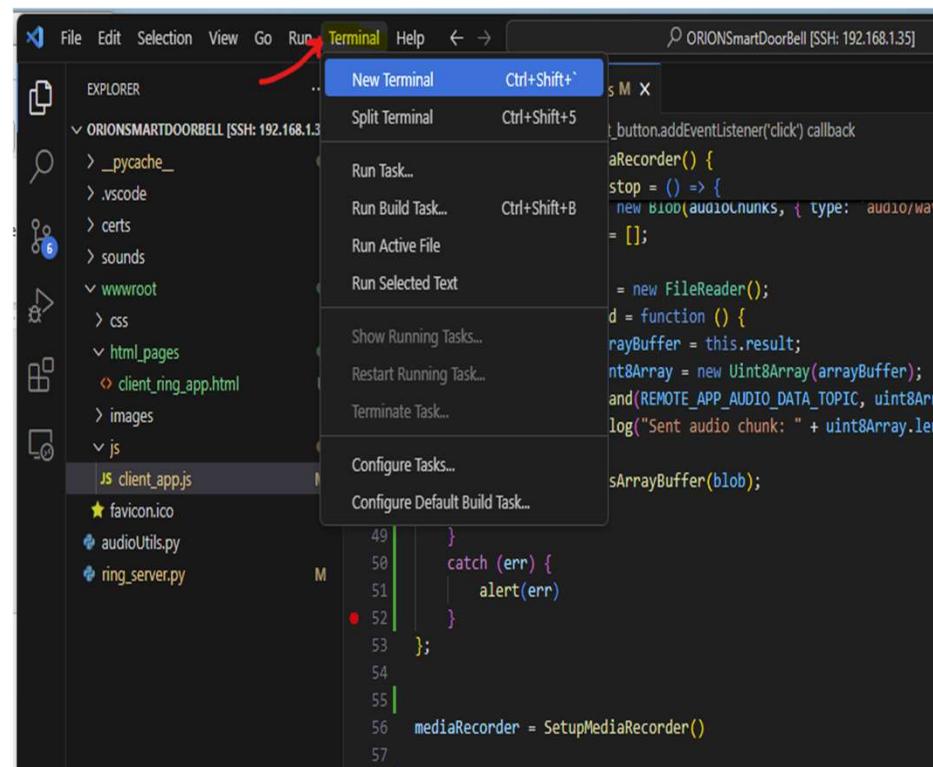


Back-end (Server) Device



<https://openai.com/>

Open New Terminal Window in VSCode



A screenshot of the Visual Studio Code interface showing a new terminal window. A large red arrow points from the previous screenshot to this one. The terminal window displays a command-line session:

```
pi@viper:~/ORIONSmartDoorBell $ pwd
/home/pi/ORIONSmartDoorBell
pi@viper:~/ORIONSmartDoorBell $
```

The code editor on the right shows the 'client_app.js' file with some code related to audio recording and sending data to a remote topic. The status bar at the bottom indicates the connection is to 'bash - ORIONSmartDoorBell'.

Enter the command `pwd`

to print the current working directory (folder)

The back-end (Python) code in VSCode: *ring_server.py*

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows project structure under "ORIONSMARTDOORBELL [SSH: 192.168.1.35]".
- Editor:** Two tabs are open: "ring_server.py" (Python) and "client_app.js" (JavaScript). The Python code handles MQTT connections, motion detection, and streaming video.
- Terminal:** A terminal window at the bottom shows the command `pi@viper:~/ORIONsmartDoorBell $ pip3 install paho-mqtt`.

Install Prerequisites Python Packages:

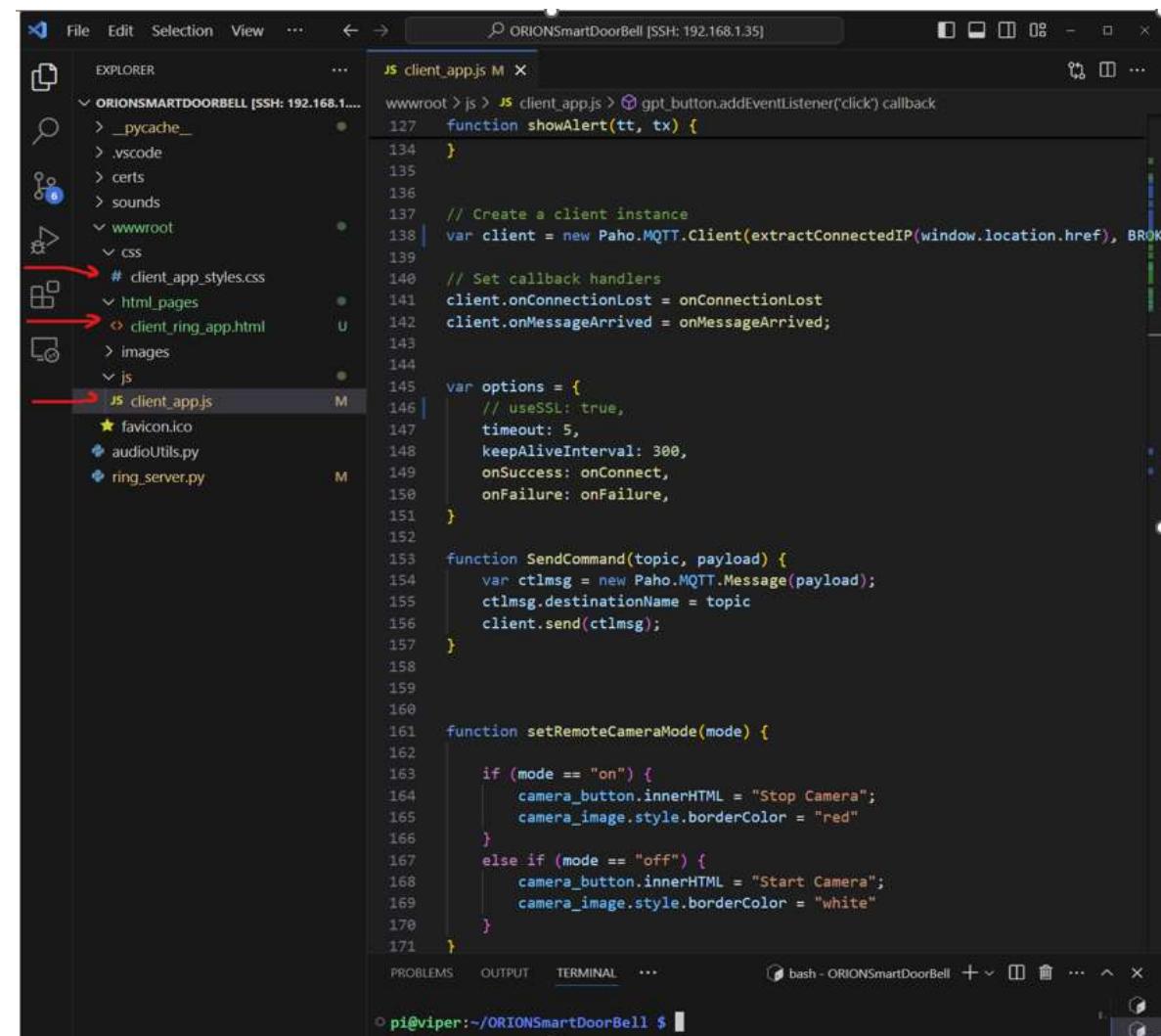
1. Controlling the Doorbell using the JavaScript client application (from PC or mobile device) requires the server to implement MQTT functionality. For MQTT support in the server install the: paho MQTT library package
Command: *pip install paho-mqtt*
 2. Allowing the person at the door to communicate (speak) via a doorbell intercom requires access to the Raspberry microphone. To read and transmit audio from the Raspberry Pi over MQTT, install audio support for the pyaudio package
Commands:
pip install pyaudio
sudo apt-get install portaudio19-dev python3-pyaudio

Let's take a few mins to review the code together

The Front-end (Web App) code is implemented using a combination of **JavaScript**, **HTML5**, and **CSS**

1. The project folder: `wwwroot` contains a subfolder `html_pages` with single HTML5 page file: `client_ring_app.html`
 - The HTML page defines the “structure and content” for the web page (the client app)
2. The subfolder `js` contains a single JavaScript code file: `client_app.js`
 - The JavaScript code implements the “behavior” for the client app
3. The subfolder `CSS` contains a single cascading style sheet: `client_app_styles.css`
Cascading style sheets define the “style (visual theme)” for the client app.

Note: the JavaScript code enables the user to control the doorbell. This is implemented using an MQTT publish/subscribe message topic strategy. Because this code runs in the context of the web browser (Chrome, Firefox, Safari etc.) it implements MQTT over Web Sockets.



The screenshot shows a terminal window titled "ORIONSmartDoorBell [SSH: 192.168.1.35]". The left pane is an "EXPLORER" view showing the directory structure:

- ORIONSMARTDOORBELL [SSH: 192.168.1.35]
- > __pycache__
- > .vscode
- > certs
- > sounds
- > wwwroot
- > css
 - # client_app_styles.css
- > html_pages
 - client_ring_app.html
- > images
- > js
 - client_app.js
- > favicon.ico
- audioUtils.py
- ring_server.py

Red arrows point from the "css", "html_pages", and "js" entries in the Explorer view to their corresponding entries in the terminal's file list. The right pane displays the content of the `client_app.js` file:

```
File Edit Selection View ... ← → ⌂ ORIONSmartDoorBell [SSH: 192.168.1.35]
JS client_app.js M X
wwwroot > js > JS client_app.js > gpt.button.addEventListener('click') callback
127 function showAlert(tt, tx) {
134 }
135
136
137 // Create a client instance
138 var client = new Paho.MQTT.Client(extractConnectedIP(window.location.href), BROK
139
140 // Set callback handlers
141 client.onConnectionLost = onConnectionLost
142 client.onMessageArrived = onMessageArrived;
143
144
145 var options = {
146   // UseSSL: true,
147   timeout: 5,
148   keepAliveInterval: 300,
149   onSuccess: onConnect,
150   onFailure: onFailure,
151 }
152
153 function SendCommand(topic, payload) {
154   var ctlmsg = new Paho.MQTT.Message(payload);
155   ctlmsg.destinationName = topic
156   client.send(ctlmsg);
157 }
158
159
160 function setRemoteCameraMode(mode) {
161
162   if (mode == "on") {
163     camera_button.innerHTML = "Stop Camera";
164     camera_image.style.borderColor = "red"
165   }
166   else if (mode == "off") {
167     camera_button.innerHTML = "Start Camera";
168     camera_image.style.borderColor = "white"
169   }
170 }
171
```

The terminal also shows the command `pi@viper:~/ORIONSmartDoorBell $`.

Supported Doorbell Server (Back-end) Modes:

The back-end server can be started in two modes: “manual” and “motion”. The command below illustrates how to start the server

`python ring_server.py --mode manual --secure off 2> /dev/null`

- Server script name: `python ring_server.py`
- Optional “mode” command line argument: `--mode manual`
 - manual** (button press) or **motion** (PIR detection)
default = **manual**
- Optional “security” command line argument: `--secure on`
 - on** configures for TLS support, **off** is clear text. Default is **off**
- Redirect error message from internal libraries: `2> /dev/null`

```
pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode motion --secure on 2> /dev/null
"Smart" Doorbell server started on port: 8001
connected over web sockets: Success
Connected to MQTT Broker on port 1883 established
Subscribed to topics
```

```
play doorbell chime
recording started      pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode manual --secure off 2> /dev/null
"Smart" Doorbell server started on port: 8000
Connected to MQTT Broker on port 1883 established
connected over web sockets: Success
Subscribed to topics
```

```
pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode manual --secure on 2> /dev/null
"Smart" Doorbell server started on port: 8001
Connected to MQTT Broker on port 1883 established
connected over web sockets: Success
Subscribed to topics
```

* Notice the port (highlighted) using
-secure on versus -secure off

View Doorbell Client Application (in Unsecure mode)

- Download and install the Brave browser on the Windows laptop
 - <https://brave.com/download/>
- Open The “Brave” browser and copy the following URL into address bar, press enter
 - [http://\[IP-address\]:8000/index.html](http://[IP-address]:8000/index.html)
 - Note: replace [IP-address] with IP address of the Raspberry Pi
 - Notice the protocol in URL is *http*

- **Questions:**

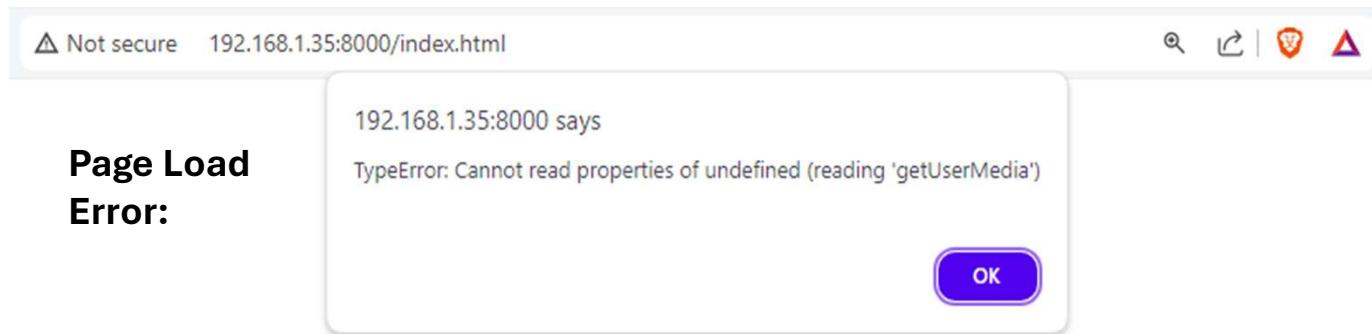
1. Record the initial result of viewing the URL in the browser? What happened?
2. Now, start the server(command using command below) in “manual” mode with security “off”



```
pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode manual --secure off 2> /dev/null
"Smart" Doorbell server started on port: 8000
Connected to MQTT Broker on port 1883 established
connected over web sockets: Success
Subscribed to topics
```

- 2.1 View The URL again (refresh by pressing Enter with cursor in address bar)
 - You should receive a type error message like the following: * **See next slide→**

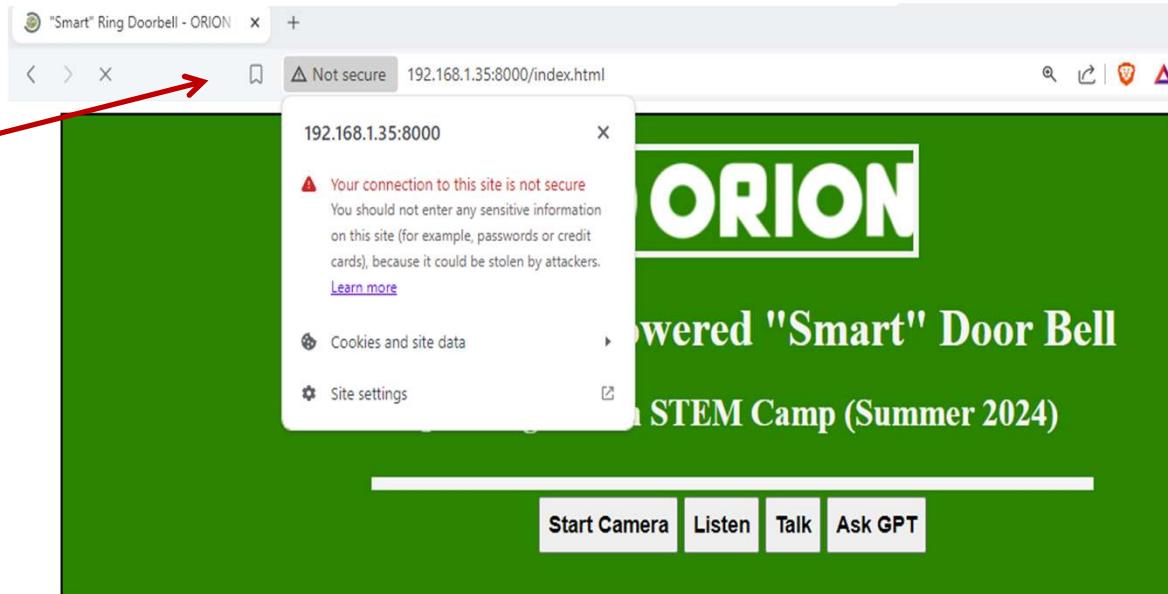
Connect to the Server using the Doorbell Client (Web) Application



**Page Load
Error:**

Press the “Ok” button and the page should Load correctly

Note: the connection is not secure



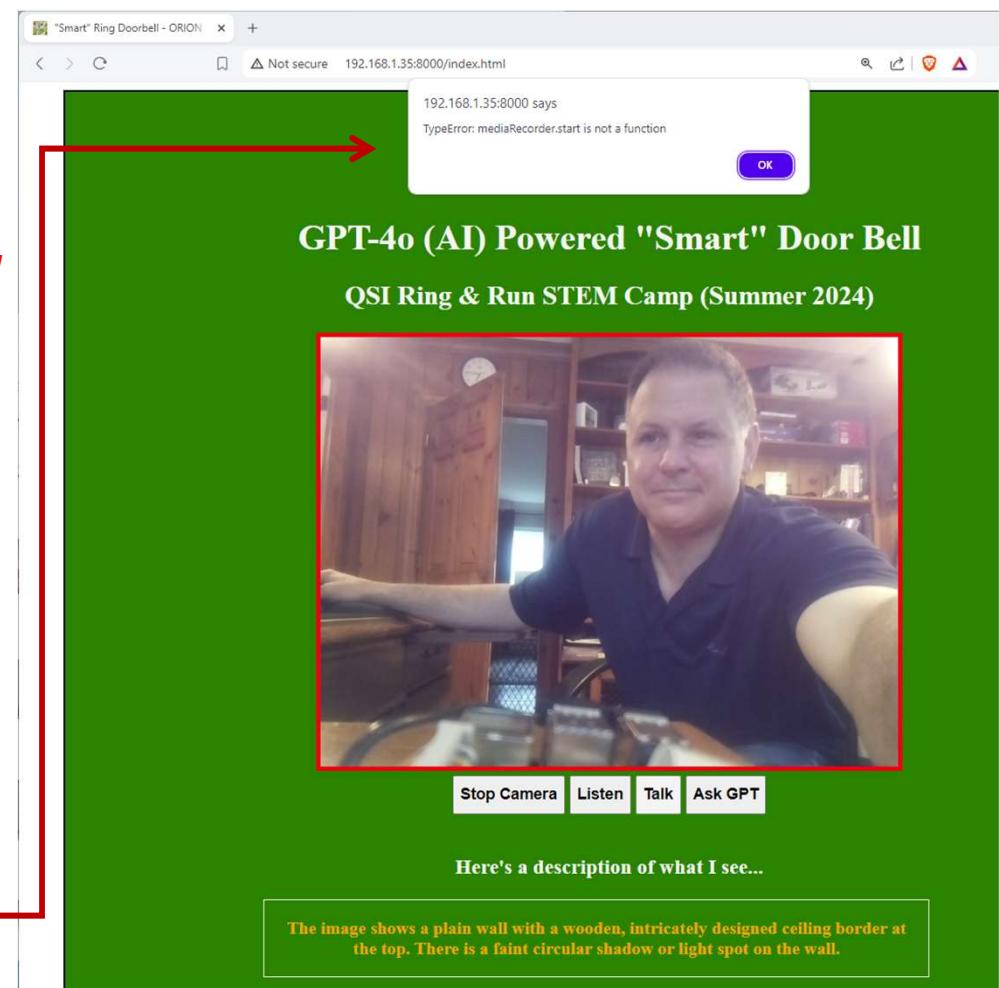
Experiment with the Doorbell Application (in Unsecure mode)

- Notice the server is exposed on TCP port 8000
- Experiment with the buttons, observe and record the behavior
 - *Note: the camera may not behave as expected on the first run. This is an unresolved bug. If this happens, then simply reload page in the browser.*



Challenge Questions

1. Click “Ask GPT” and record the result
2. When the page first loads, and whenever you click “Talk” a JavaScript error (related to Page Load error illustrated on the previous slide) is displayed
 - Why do you do these errors occur?



Note: press F12 in the Browser for “Developer Tools”, click “Console” to see messages

Answer to Challenge Question (unsecure mode):

- If you said something to related to not using **proper security practices**, then you would be correct!
- Modern Web browser (security) policies will not allow JavaScript code to access certain browser managed media devices (such microphones) over unsecure HTTP connections

This is answered directly in the [MDN documentation for `MediaDevices.getUserMedia\(\)`](#):

24

Note: If the current document isn't loaded securely, `navigator.mediaDevices` will be `undefined`, and you cannot use `getUserMedia()`. See [Security](#) for more information on this and other security issues related to using `getUserMedia()`.

AppSimulator.net is not presented securely, so this call will always fail in the manner you observed. Serve the page over HTTPS.

Stack Overflow Post



MODULE 4: IMPLEMENT SSL/TLS

RING & RUN STEM EVENT.



ORION
OPEN ARCHITECTURE RESILIENT IOT
FOR OPERATIONAL NETWORKS

SHORT INTERLUDE: PKI AND TLS EXPLAINED

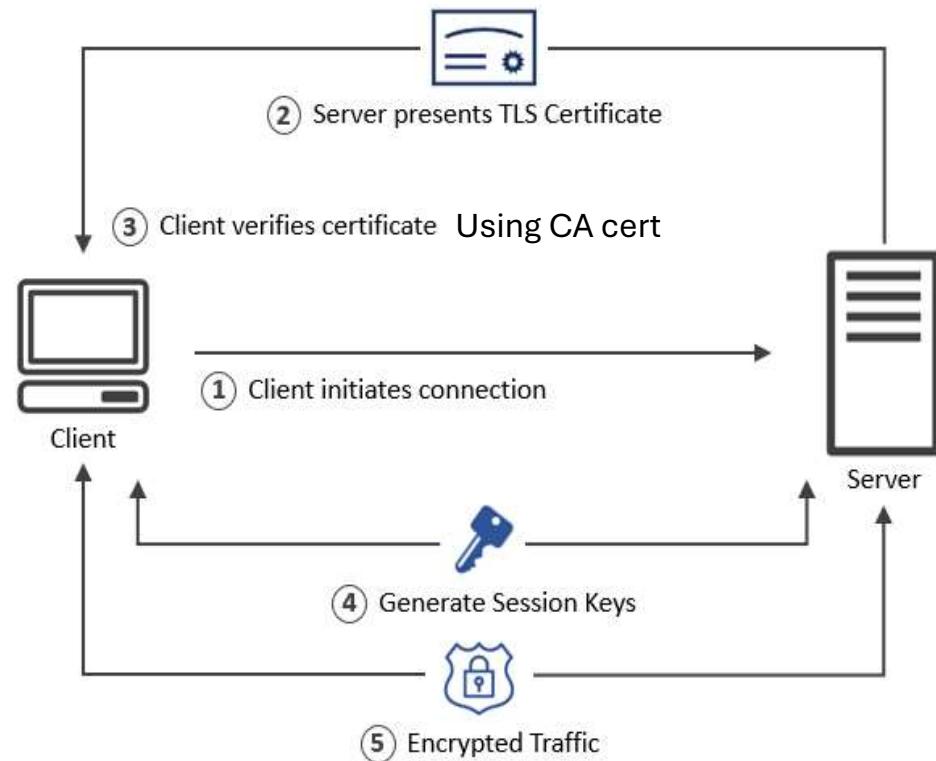
- What is PKI (Public key infrastructure) ?
 - <https://www.youtube.com/watch?v=uVaUgrxjMe0>
- What is SSL/TLS (HTTPS)?
 - <https://www.youtube.com/watch?v=j9QmMEWmcfo>

TLS (Concept) Illustrated

Example Usage:

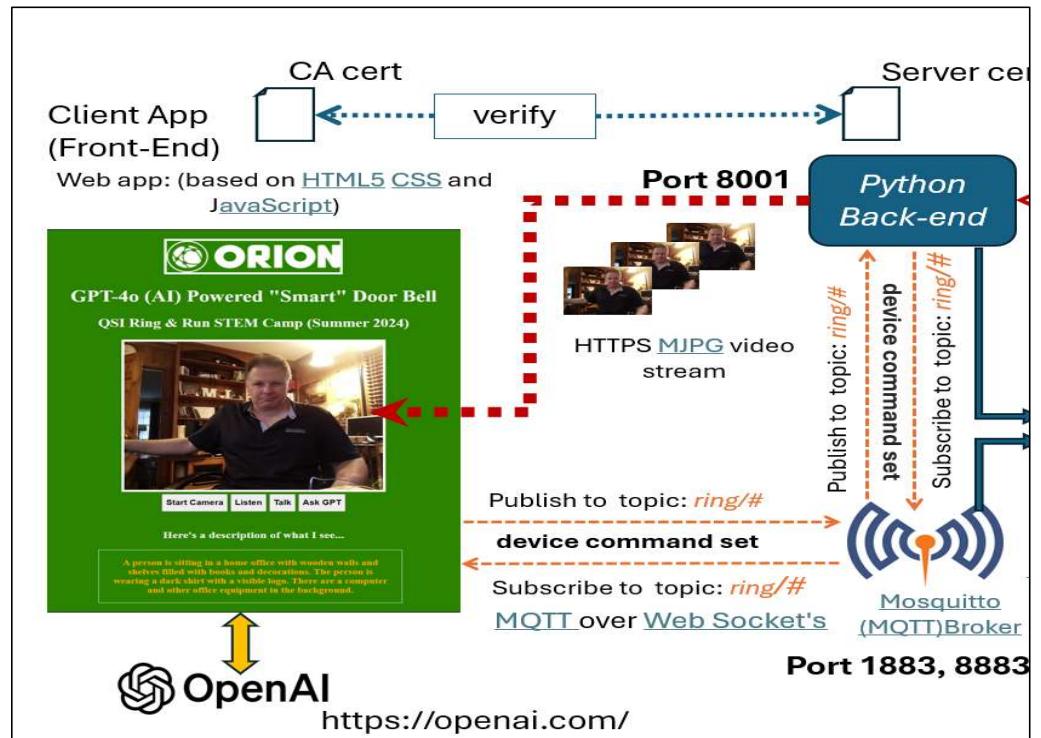
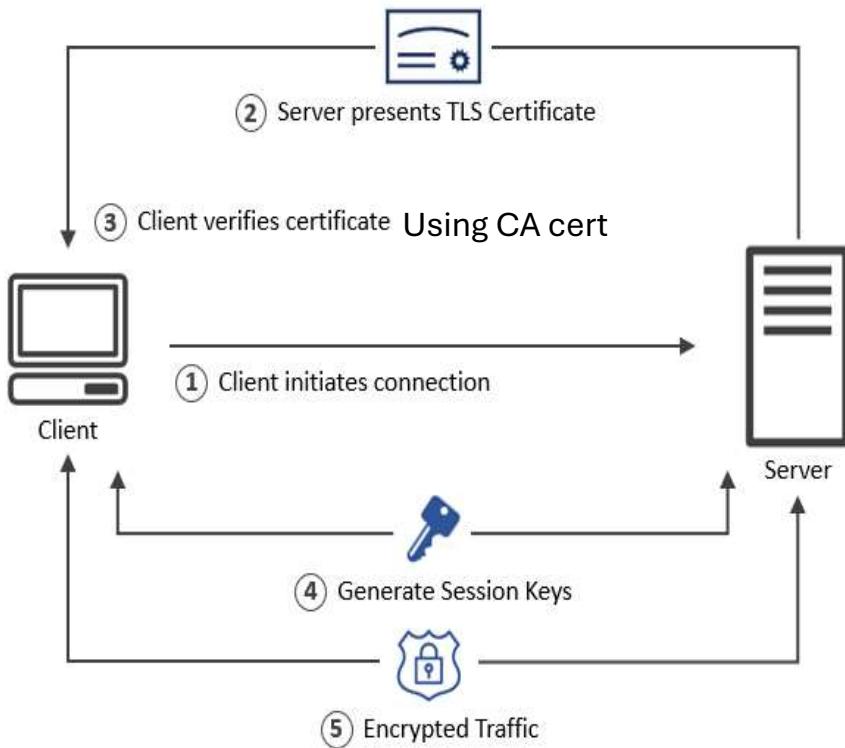
- A typical web browser and web server connection using **SSL/TLS (HTTPS)**:
 - This connection is used on the Internet to send email in Gmail, and when doing online banking, shopping etc.
 1. Browser connects to server using https.
 2. Server presents certificate to the client (Browser)
 3. Browser verifies the certificate by checking the signature of the CA. To do this the **CA certificate** needs to be in the browser's trusted store (See later)
 4. Browser uses this Public Key to agree a **session key** with the server.
 5. Web Browser and server encrypt data over the connection using the **session key**.

Source: <http://www.steves-internet-guide.com/ssl-certificates-explained/>



Source: <https://www.csa.gov.sg/Tips-Resource/internet-hygiene-portal/information-resources/tls>

Implementing TLS into the “Smart” Doorbell Design



Source: <https://www.csa.gov.sg/Tips-Resource/internet-hygiene-portal/information-resources/tls>

- Steve's guide links:
 - <http://www.steves-internet-guide.com/mosquitto-tls/>
 - <http://www.steves-internet-guide.com/ssl-certificates-explained/>

Implementing TLS into “Smart” Doorbell Design (Process Overview)

1. Generate the organization's certificate request (CSR), and purchase an organization's certificate from a Trusted CA (Verisign, GoDaddy etc.)
 - List of Trusted Certifying Authorities
 - https://developer.visa.com/pages/trusted_certifyingAuthorities

Or

2. Implement all the PKI requirements yourself using the [OpenSSL](#) toolset
 1. Generate CA (public/private) key pair and Certificate for signing and verification
 2. Generate the organization's (public/private) key pair, and CSR
 3. Verify/sign the CSR with the **organization's** CA (private key) to generate a signed server certificate (in this case for the Doorbell application)
 4. Implement TLS in organizations services (.e.g. the Doorbell) and load the signed certificate.
 5. Add the **organization's** CA to Browsers “Trusted Store”

Step 1 of 5: Create the Key Pair For Certificate Authority (CA) Certificate

- VNC to the Raspberry Pi and navigate the project sub folder, named “certs”.

```
pi@viper: ~/ORIONSmartDoorBell/  
File Edit Tabs Help  
pi@viper:~ $ cd ~/ORIONSmartDoorBell/certs/  
pi@viper:~/ORIONSmartDoorBell/certs $ █
```

- Create a key pair for your personal ORION CA: *openssl genrsa -des3 -out orion_ca.key 2048*

```
pi@viper: ~/ORIONSmartDoorBell/certs  
File Edit Tabs Help  
pi@viper:~/ORIONSmartDoorBell/certs $ openssl genrsa -des3 -out orion_ca.key 2048  
Generating RSA private key, 2048 bit long modulus (2 primes)  
.....+++++  
.....+++++  
e is 65537 (0x010001)  
Enter pass phrase for orion_ca.key:  
Verifying - Enter pass phrase for orion_ca.key:  
pi@viper:~/ORIONSmartDoorBell/certs $ █
```

Note: Use the pass phrase “ORION” (to keep things simple)

Step 2 of 5: Create the CA Certificate

- Create a certificate for the CA using the **orion_ca.key** that we just created:

```
openssl req -new -x509 -days 1826 -key orion_ca.key -out orion_ca.crt
```

```
pi@viper:~/ORIONsmartDoorBell/certs $ openssl req -new -x509 -days 1826 -key orion_ca.key -out orion_ca.crt
Enter pass phrase for orion_ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ORION
Organizational Unit Name (eg, section) []:IoT Lab
Common Name (e.g. server FQDN or YOUR name) []:ORION.org
Email Address []:admin@orion.org
-----
```

Note: You will be prompted to enter information that will be incorporated into the CA cert. Use the highlighted values in the above example

Step 3 of 5: Create the Key Pair (Back-end) Server

Create a key pair for the server : *openssl genrsa -out ring_server.key 2048*

```
pi@viper:~/ORIONSmartDoorBell/certs $ openssl genrsa -out ring_server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
pi@viper:~/ORIONSmartDoorBell/certs $
```

Note: This key pair will be used to create the Server Certificate Request (CSR) in the next step

Step 4 of 5: Create Server Certificate Request (CSR)

- The CSR contains an organization's public keys, and domain identity (the common name)
 - The domain identity is often the fully qualified domain name (FQDN) of the requesting organization (for example: www.mydomain.com)
- The CSR is sent to (a trusted) CA that verifies the CSR (domain ownership etc.)
 - The CA signs CSR with its private key to generate the certificate which is then sent back to requesting organization.
 - The CA signature (verifiably) binds the requestor's identity (the FQDN / SAN) to its public key
 - The CA certificate in (the CA chain containing the CA's public key) is publicly available, and used to verify the signature of any certificates signed by that CA
 - Trusted CA certificates are preinstalled into the "trusted store" of most web browsers.
 - This is how the browser verifies the identity of the server it is connected to (.e.g. your Bank)

Step 4 of 5: Create Server Certificate Request (CSR) cont.

- In the “certs” subfolder, open ***san.cnf*** with editor (nano or vi), update the file (as shown) with the IP address and hostname of your Raspberry Pi. Save and exit the file
- Create the CSR with the following command:

```
openssl req -new -out ring_server.csr -key ring_server.key -config san.cnf
```

```
File Edit Tabs Help
pi@viper:~/ORIONSmartDoorBell/certs $ openssl req -new -out ring_server.csr -key ring_server.key -config san.cnf
pi@viper:~/ORIONSmartDoorBell/certs $
```

- When filling out the CSR form, the **common name** is important piece that is verified, and is usually the **domain name** of the server/organization

- Note: for simplicity reasons, we will **use the IP address of the Raspberry Pi as the common name**. Some browsers are configured to require SAN (subject alternative names) so that will be specified using an external configuration file located in the “certs” subfolder

```
[ req ]
default_bits      = 2048
distinguished_name = req_distinguished_name
req_extensions    = req_ext
prompt            = no

[ req_distinguished_name ]
C      = US
ST     = New York
L      = Rome
O      = ORION
OU    = IoT
CN    = 192.168.1.35

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
IP.1 = 192.168.1.35
DNS.2 = viper
```

Step 5 of 5: Generate the Server Certificate

- Use our **ORION_CA key** to verify and sign the server certificate. The command below creates the requesting organization's(the server) certificate: **ring_server.crt**
 - Notice the CSR from Step 4 is using the CA key file to verify and sign *ring_server.crt*
openssl x509 -req -in ring_server.csr -CA orion_ca.crt -CAkey orion_ca.key -CAcreateserial -out ring_server.crt -days 365 -extensions req_ext -extfile san.cnf

```
pi@viper:~/ORIONsmartDoorBell/certs $ openssl x509 -req -in ring_server.csr -CA orion_ca.crt -  
CAkey orion_ca.key -CAcreateserial -out ring_server.crt -days 365 -extensions req_ext -extfile  
san.cnf  
Signature ok ←  
subject=C = US, ST = New York, L = Rome, O = ORION, OU = IoT, CN = 192.168.1.35  
Getting CA Private Key  
Enter pass phrase for orion_ca.key:  
pi@viper:~/ORIONsmartDoorBell/certs $
```

- Note: Our ORION organization is acting as the local CA. This is just as secure as an official (trusted) CA. If ORION maintains a small number of stakeholders, it can just as easily distribute the CA certificate to stakeholders to enable verifiable transactions
 - The issue becomes untenable when broad acceptance is required
 - Anyone can initiate transactions (e.g., ecommerce)
 - In which case, you can't distribute your cert

Congratulations! PKI Requirements Complete

Enter command: `ls -l`

You output should resemble the following:

```
pi@viper:~/ORIONSmartDoorBell/certs$ ls -l
total 28
-rw-r--r-- 1 pi pi 1424 Jul 11 10:00 orion_ca.crt
-rw----- 1 pi pi 1743 Jul 11 09:57 orion_ca.key
-rw-r--r-- 1 pi pi    41 Jul 11 10:01 orion_ca.srl
-rw-r--r-- 1 pi pi 1298 Jul 11 10:01 ring_server.crt
-rw-r--r-- 1 pi pi 1054 Jul 11 10:01 ring_server.csr
-rw----- 1 pi pi 1675 Jul 11 10:00 ring_server.key
-rw-r--r-- 1 pi pi   319 Jul 10 10:24 san.cnf
```

- Summary of Activities:

- Generate CA key pair
- Generate certificate for the CA using the **CA key**
- Generate doorbell server (back-end) key pair
- Create a certificate request (**CSR**)
 - Public key and domain name identity
- Send CSR to CA Authority for signing and returned certificate
 - Cert binds and identity (domain) to the public key
 - Example: passport binds a picture of person to a name etc. (identity)
- Client of the organization's services, can verify the identity of the organization using CA cert to verify the organization's certificate (signature verification)

View the Doorbell Application (in Secure mode)

- Start the server (using command below) in “manual” mode with security “on”

```
pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode manual --secure on 2> /dev/null
"Smart" Doorbell server started on port: 8001
Connected to MQTT Broker on port 1883 established
connected over web sockets: Success
Subscribed to topics
```

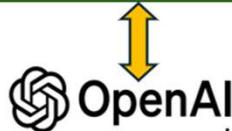
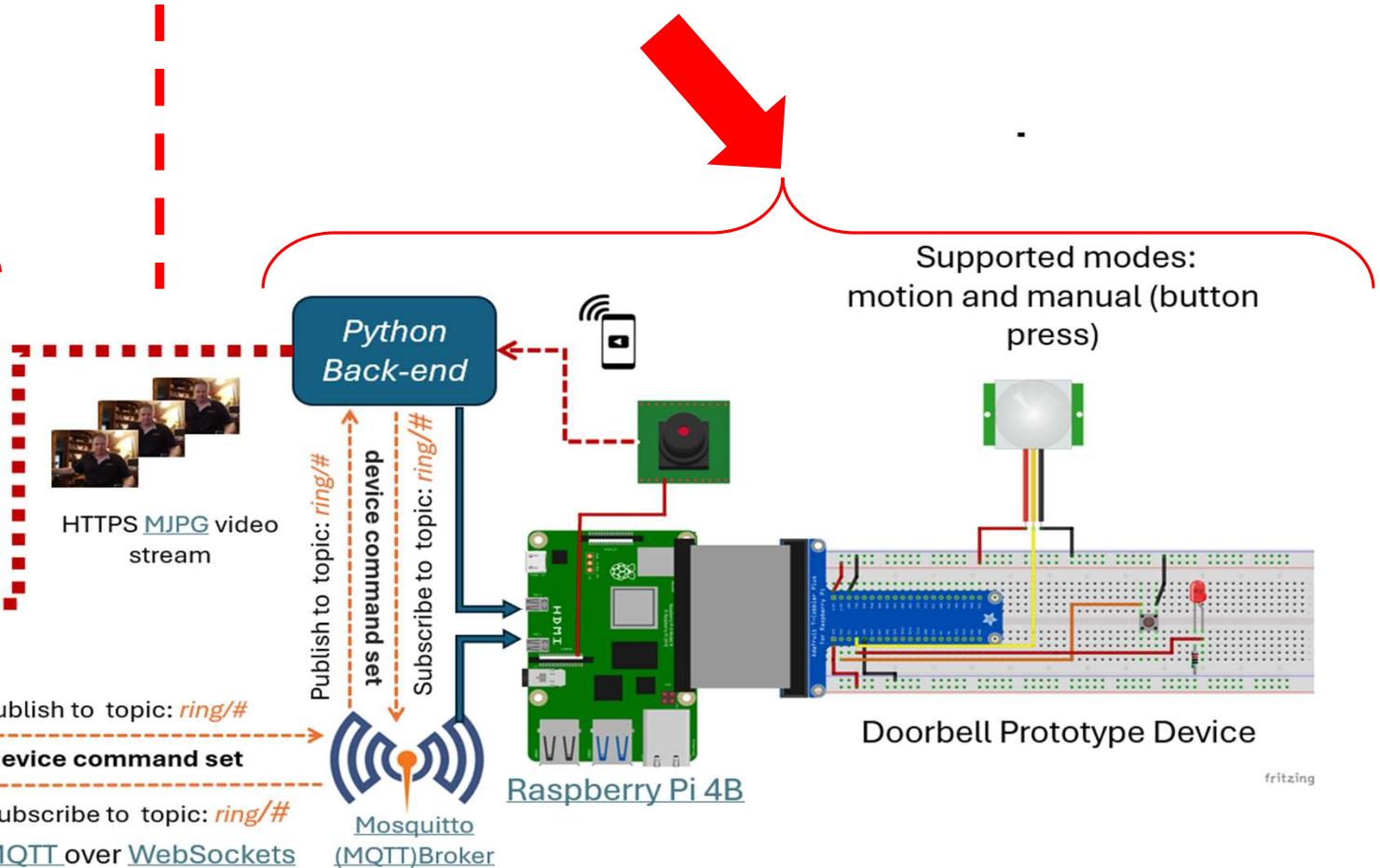
- *Notice: server port number is 8001 (not 8000 as is the case with insecure mode)*

- Open The “Brave” browser and copy the following URL into address bar, press enter
 - [https://\[IP-address\]:8001/index.html](https://[IP-address]:8001/index.html)
 - *Replace [IP-address] with IP address of the Raspberry Pi*
 - *Notice the protocol in URL has changed to https*

Front-end (web) App

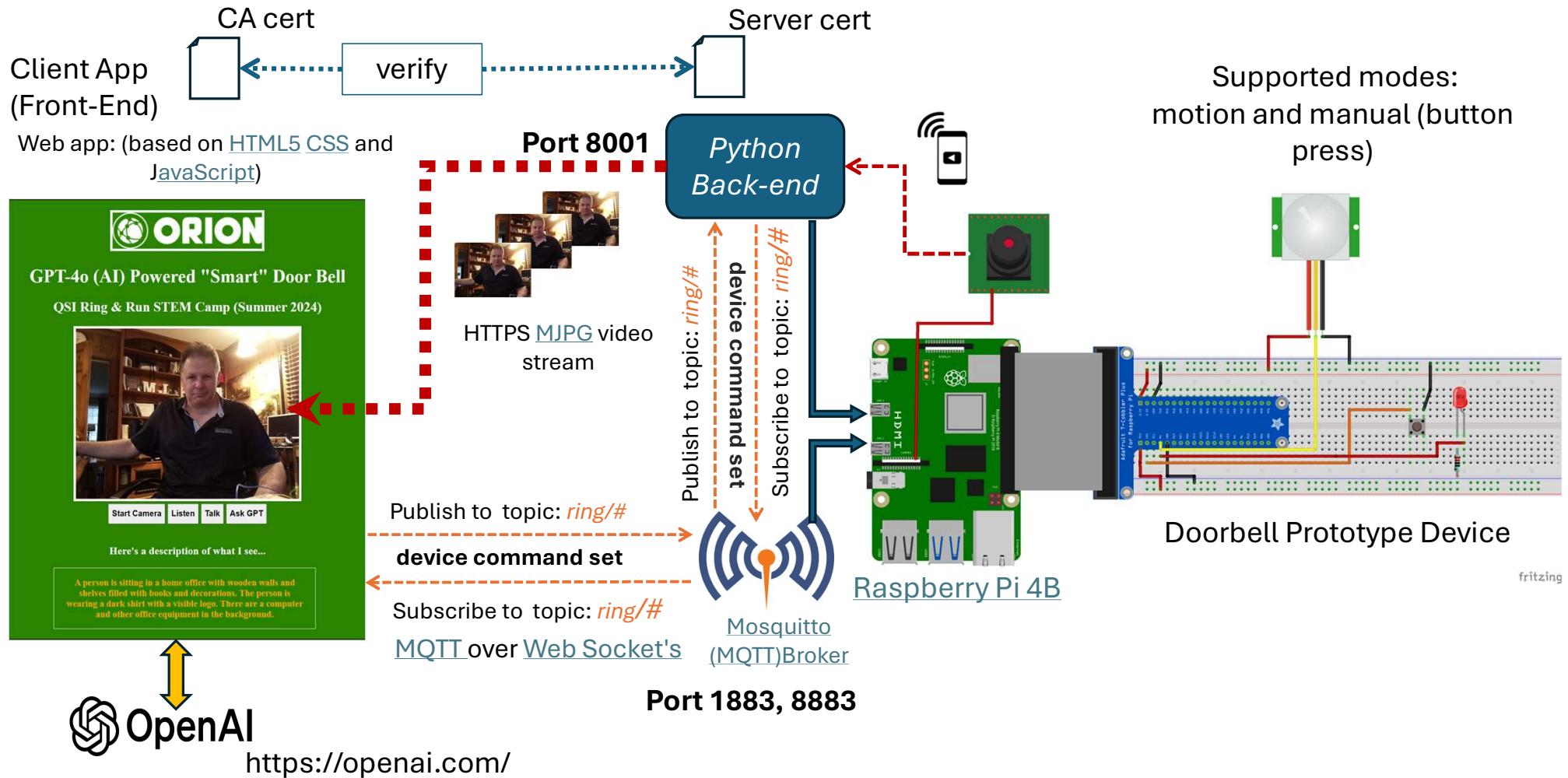


Back-end (Server) Device



<https://openai.com/>

IoT enabled “Smart” Doorbell Concept with TLS



Viewing the Doorbell Application (Secure mode)

- URL:

- **https://[IP-address]:8001/index.html**
 - Replace [IP-address] with IP address of the Raspberry Pi
 - Notice the protocol in URL has changed to **https** and exposed port 8001



Challenge Questions:

1. What is the browser indicating?
2. How would we rectify this error?

Not secure https://192.168.1.35:8001/index.html

192.168.1.35:8001

Your connection to this site is not secure
You should not enter any sensitive information on this site (for example, passwords or credit cards), because it could be stolen by attackers.
[Learn more](#)

Certificate is not valid

Cookies and site data

Site settings

Your connection is not private
Attackers might be trying to steal your information from 192.168.1.35 (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

Hide advanced

Back to safety

This server could not prove that it is 192.168.1.35; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Answer to Challenge Question (secure mode):

- ...Certificate Error!
 - missing (valid) PKI certificates to verify the identity of server and establish an encrypted session between the client and the server
 - How do we rectify these issues?
 - ✓ Add the CA cert to Brower's "Trusted Store"

Adding the ORION CA Certificate to the Browser Trusted Store

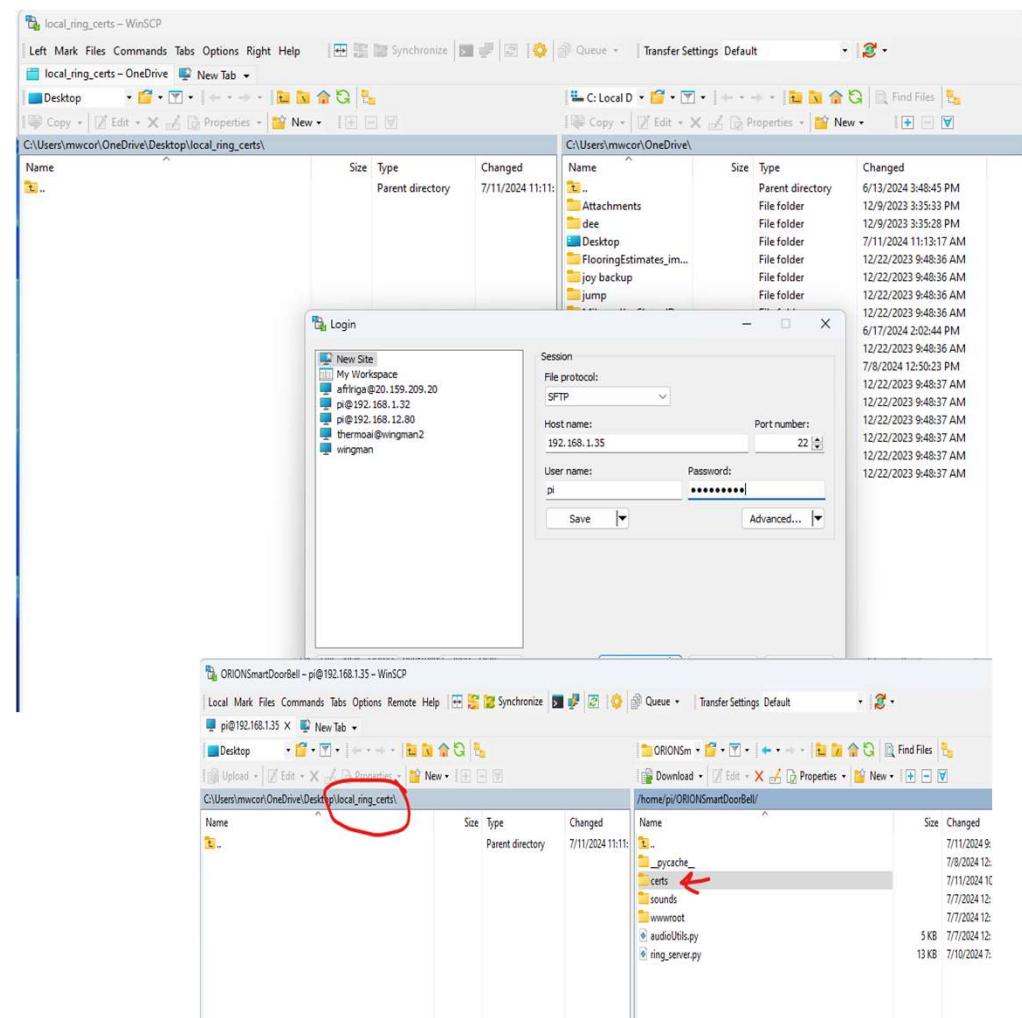
- The local (ORION CA) signed the ring server certificate with it's private key
 - When a user connects to the ring server using a web browser, the browser can verify the identity of the server by verifying the signature in the server's certificate
- The CA certificate contains the CA's public key, (and thanks to PKI asymmetric encryption), the browser can use the CA public key to very signature generated with the CA's private key.
- To accomplish this, we begin by adding the ORION CA certificate to the browser's "Trusted Store" - a folder on the device where certificates are stored/managed
 - We will accomplish this with "Brave "browser that we previously installed on the laptop.

Use WinSCP to copy the Certs Folder

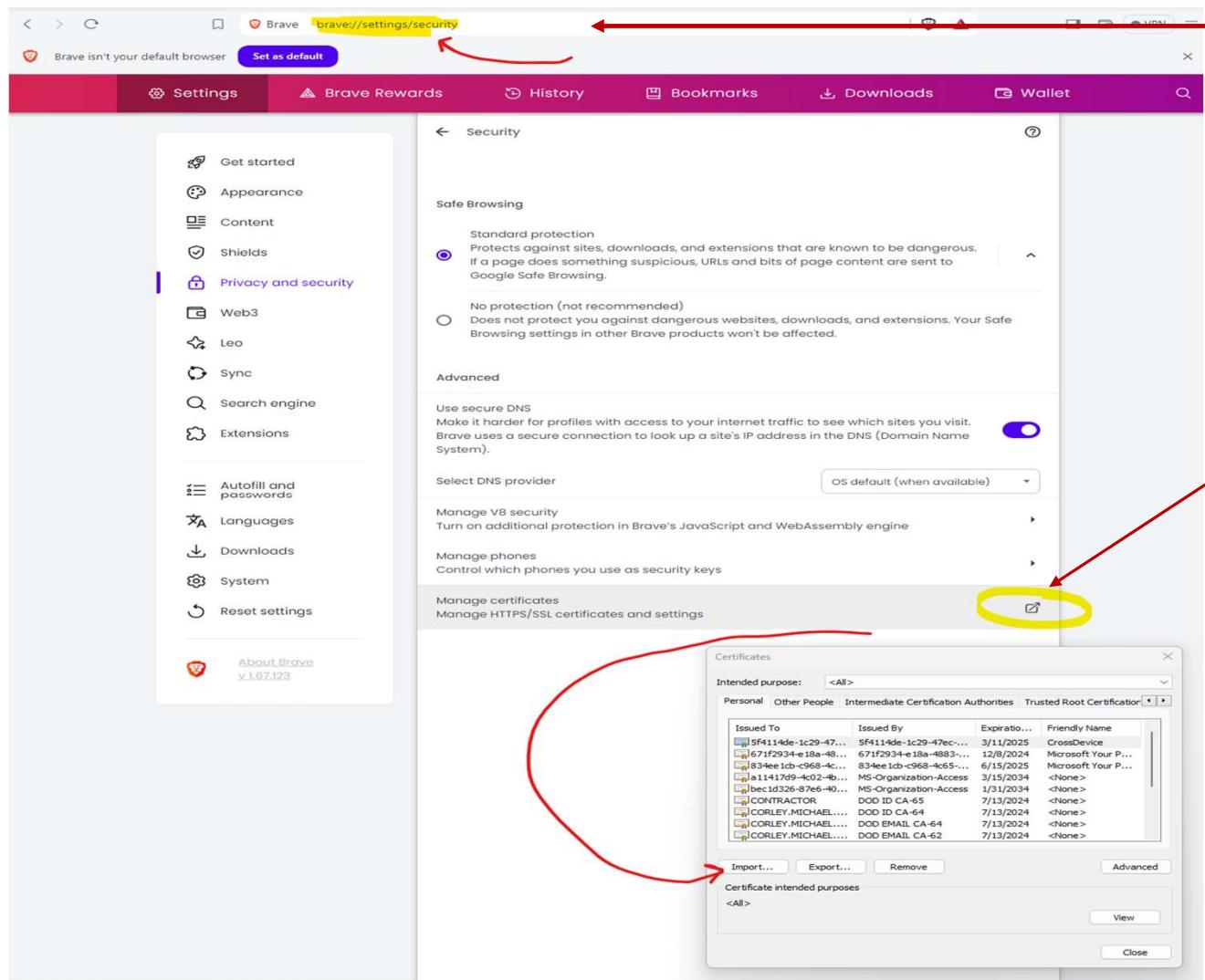
- We need the certificates (currently stored on the Raspberry Pi) copied to the Windows laptop so we can import the ORION CA cert into the Brave browser's "Trusted Store" location:

- For this we will use WinSCP
 - Download WinSCP here:
<https://winscp.net/eng/download.php>

- Create a folder on the Windows 11 Desktop, named: "[local_ring_certs](#)"
- Open WinSCP, connect to the Raspberry Pi, and copy the project subfolder "[certs](#)" to the "[local_ring_certs](#)" folder on the Windows Desktop



Adding the ORION CA Certificate to the Brave Trusted Store (cont.)

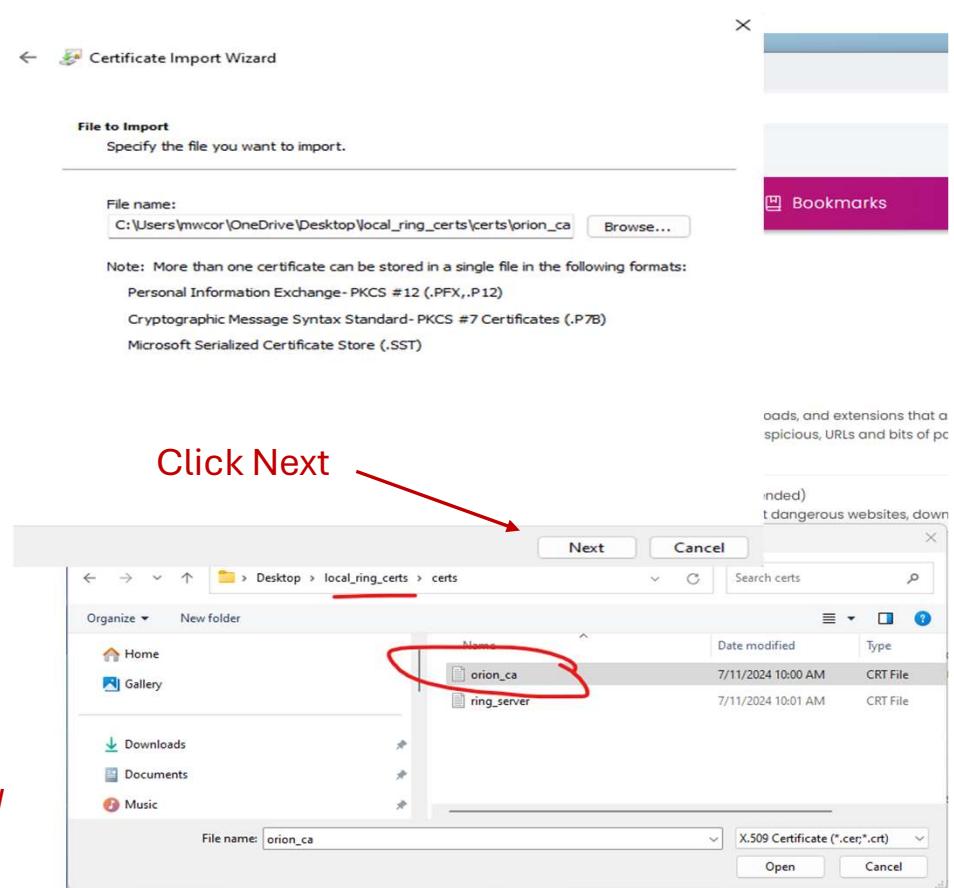
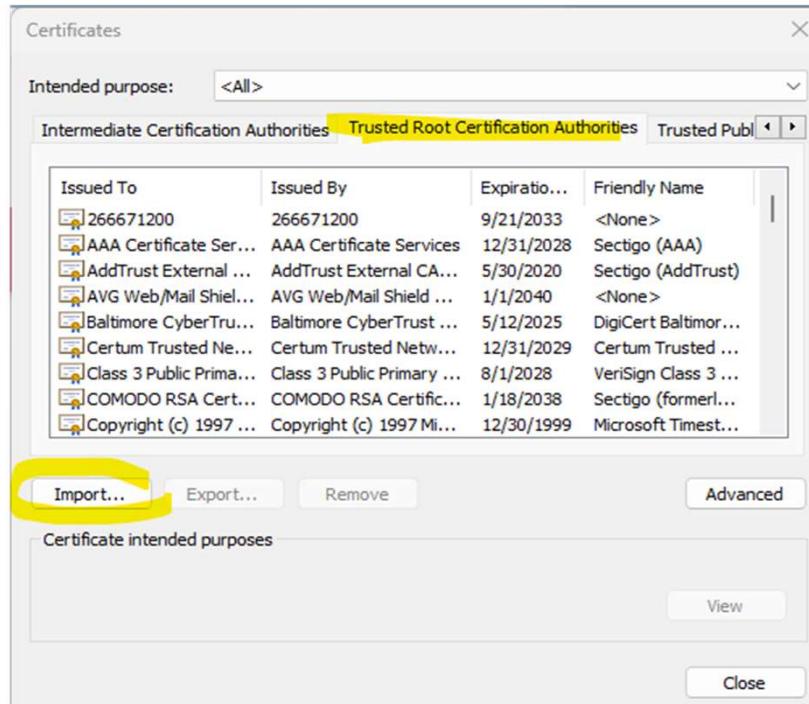


1. Open Brave browser and enter the following in the address bar: *brave://settings/security*

2. Click “Manage Certificates”

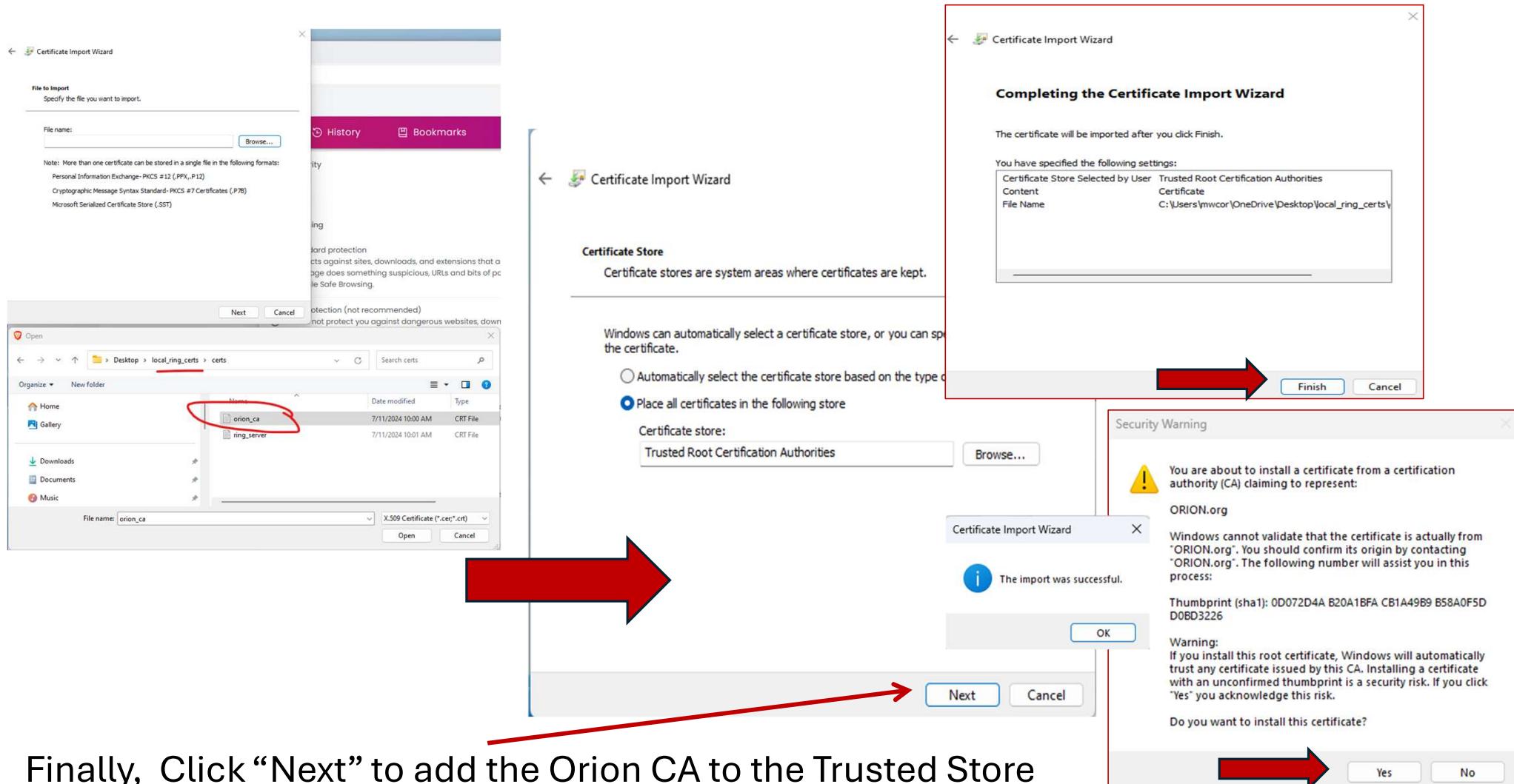
3. See next slide

Adding the ORION CA Certificate to the Brave Trusted Store (cont.)

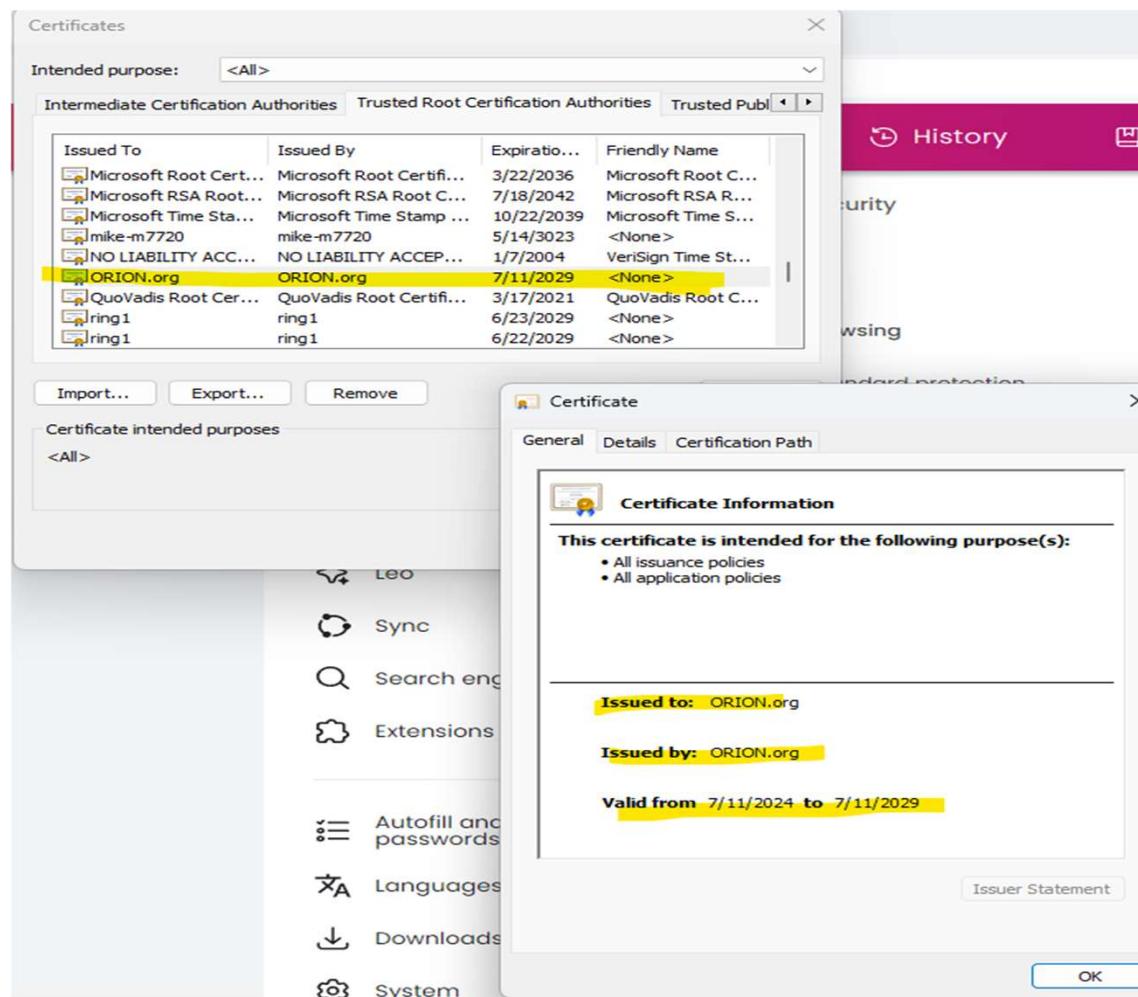


1. Click “Trusted Root Certifications” Tab
 - Notice the official list of Trusted CA certs displayed
2. Click “Import” and navigate to “local_ring_certs” folder on the Windows Desktop, select “orion_ca.crt” and click “Open”

Adding the ORION CA Certificate to the Brave Trusted Store (cont.)



Congratulations, You've Added the ORION Trusted CA Certificate



View the Doorbell Application (Secure mode)

- Start the server(using command below) in “manual” mode with security “on”

```
pi@viper:~/ORIONSmartDoorBell $ python ring_server.py --mode manual --secure on 2> /dev/null
"Smart" Doorbell server started on port: 8001
Connected to MQTT Broker on port 1883 established
connected over web sockets: Success
Subscribed to topics
```

- *Notice: server TCP port number is 8001 (not 8000 as is the case with insecure mode)*

- Open The “Brave” browser and copy the following URL into address bar, press enter

- [https://\[IP-address\]:8001/index.html](https://[IP-address]:8001/index.html)
 - *Replace [IP-address] with IP address of the Raspberry Pi*
 - *Notice the protocol in URL has changed to https*
 - *Notice the TCP port in request has changed to 8001*

View the Doorbell Application(Secure mode): With PKI Certs. installed

Open Brave and enter following URL into the address bar: [https://\[IP-address\]:8001/index.html](https://[IP-address]:8001/index.html)
Replace [IP-address], with IP address of your Raspberry Pi

The certificate error observed in slide 101, has been successfully corrected, however, viewing the **console** output in **developer tools (F12)** we have a different error.



Challenge Question: What does this error message indicate ?

- Hint: View the Mosquitto Broker configuration file: **nano /etc/mosquito/mosquitto.cnf**

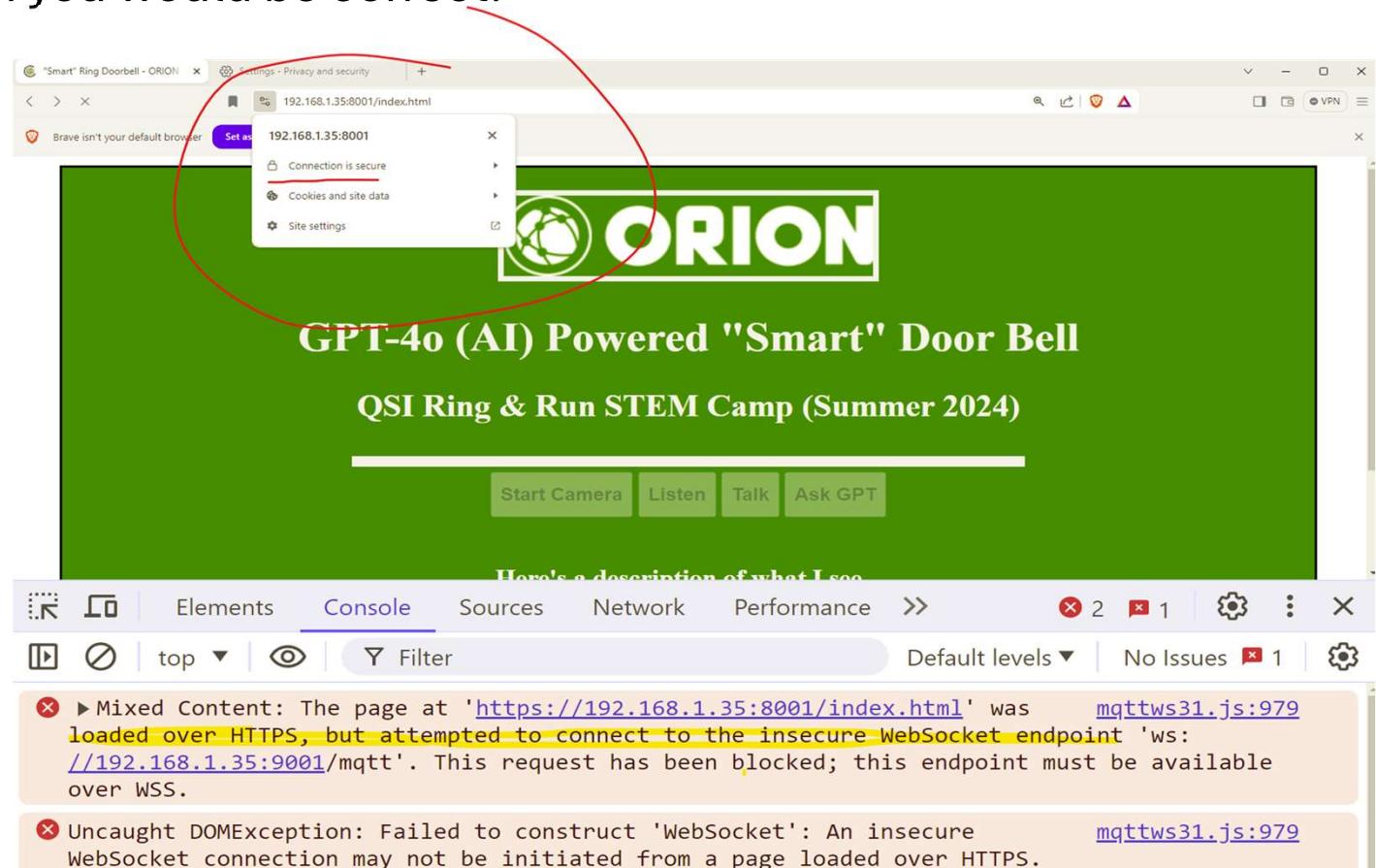
A screenshot of a Brave browser window showing a secure connection to a "Smart Ring Doorbell - ORION" application. The address bar shows "192.168.1.35:8001/index.html". A context menu is open over the address bar with options: "Set as default browser", "Connection is secure", "Cookies and site data", and "Site settings". A red arrow points from the text above to this menu. The main content area displays the "ORION" logo and the text "GPT-4o (AI) Powered 'Smart' Door Bell" and "QSI Ring & Run STEM Camp (Summer 2024)". Below the content is a toolbar with "Start Camera", "Listen", "Talk", and "Ask GPT" buttons. At the bottom, the developer tools console tab is active, showing two error messages:

- Mixed Content: The page at '<https://192.168.1.35:8001/index.html>' was loaded over HTTPS, but attempted to connect to the insecure WebSocket endpoint 'ws://192.168.1.35:9001/mqtt'. This request has been blocked; this endpoint must be available over WSS.
- Uncaught DOMException: Failed to construct 'WebSocket': An insecure WebSocket connection may not be initiated from a page loaded over HTTPS.

Viewing the Doorbell (Secure mode): With PKI Certs. Installed (cont.)

Answer: If you said the Mosquitto MQTT Broker is not currently configured for HTTPS/TLS support, then you would be correct!

Let's update Mosquitto MQTT Broker configuration!



Update the MQTT Broker Configuration (see Slide 69 for the current configuration)

- From VNC session, or from the VSCode terminal, and open the mosquito broker configuration file: `sudo nano /etc/mosquitto/mosquitto.conf`
 - The current configuration file should resemble the screen shot below.
- To update the Broker configuration for compliance with TLS, we need to add another “listener” endpoint that uses the certificates we created previously.
- Append following lines to the end of the mosquitto.conf file:
 - listener 8883
 - protocol websockets
 - cafile /etc/mosquitto/certs/orion_ca.crt
 - keyfile /etc/mosquitto/certs/ring_server.key
 - certfile /etc/mosquitto/certs/ring_server.crt
- The updated mosquitto.conf should resemble the screen shot below:

```
pi@viperpi: ~
File Edit Tabs Help
GNU nano 5.4          /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
listener 9001
protocol websockets
allow_anonymous true
```

```
GNU nano 5.4          /etc/mosquitto/mosquitto.conf
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
listener 9001
protocol websockets
allow_anonymous true

listener 8883
protocol websockets
cafile /etc/mosquitto/certs/orion_ca.crt
keyfile /etc/mosquitto/certs/ring_server.key
certfile /etc/mosquitto/certs/ring_server.crt

# note: you can specify details such as the TLS version and supported ciphers, but
# we allow the broker to use the default (and likely) the optimal configuration
```

Explanation of the Updated Broker Configuration

- listener – configures the Broker (server) to “listen” TCP/MQTT connections on TCP port 8883
- websockets – the ideal choice to enable the web browser (JavaScript client) to participate in lightweight, real-time (bidirectional) MQTT communication with an external MQTT Broker/Server
- The presence of the certificate files: cafile, keyfile, and certfile tells the Broker to support TLS on *listener* exposed on port 8883
- * **We need to copy the certificate files from the doorbell project “certs” subfolder to “certs’ subfolder of the MQTT Broker: /etc/mosquitto/certs**

```
GNU nano 5.4                                         /etc/mosquitto/mosquitto.conf
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
listener 9001
protocol websockets
allow_anonymous true

listener 8883
protocol websockets
cafile /etc/mosquittoc/certs/orion_ca.crt
keyfile /etc/mosquitto/certs/ring_server.key
certfile /etc/mosquitto/certs/ring_server.crt
# note: you can specify details such as the TLS version and supported ciphers, but
# we allow the broker to use the default (and likely) the optimal configuration
```

Copy the Certificate files to the Broker “certs” subfolder

- From VNC session terminal (or from the VSCode terminal), navigate to the doorbell project “certs” subfolder and copy certificate files to Broker configuration folder with the following command:

```
sudo cp orion_ca.crt ring_server.key ring_server.crt /etc/mosquitto/certs/
```

Your output should resemble that in the following screenshot:

```
pi@viper:~/ORIONSmartDoorBell/certs $ sudo cp orion_ca.crt ring_server.key ring_server.crt /etc/mosquitto/certs/
pi@viper:~/ORIONSmartDoorBell/certs $ ls /etc/mosquitto/certs/
orion_ca.crt  README  ring_server.crt  ring_server.key
pi@viper:~/ORIONSmartDoorBell/certs $ ls -l /etc/mosquitto/certs/
total 16
-rw-r--r-- 1 root root 1424 Jul 11 14:34 orion_ca.crt
-rw-r--r-- 1 root root 130 Sep 30 2023 README
-rw-r--r-- 1 root root 1298 Jul 11 14:34 ring_server.crt
-rw----- 1 root root 1675 Jul 11 14:34 ring_server.key
```



Challenge Question: are the above file permissions adequate for securing for certificates and key files ? Describe Why or why not? (Group answer to follow)

Restart the Broker to apply the update service: *sudo systemctl restart mosquitto*



Uh-Oh...The restart command returned an error?
Why?

Hint: consider the file ownership as shown in `ls` command output above

Restart the Broker service with the Updated Configuration:

- The Broker restart command produced an error message similar to the following:

```
pi@viper:~/ORIONSmartDoorBell/certs $ sudo systemctl restart mosquitto
Job for mosquitto.service failed because the control process exited with error code.
See "systemctl status mosquitto.service" and "journalctl -xe" for details.
```

- Notice from the output of the `ls` command below, that the files are all owned by the “root” user account
 - We copied them using `sudo` which assumes “root” user privileges
 - mosquitto process (executed as the “mosquitto” user) is attempting to the “read” these files.
 - The *server key file* which rightly set so only the owner and read/write it.
 - Because it’s owned by root, the mosquitto process cannot read it.

```
pi@viper:~/ORIONSmartDoorBell/certs $ sudo cp orion_ca.crt ring_server.key ring_server.crt /etc/mosquitto/certs/
pi@viper:~/ORIONSmartDoorBell/certs $ ls /etc/mosquitto/certs/
orion_ca.crt README ring_server.crt ring_server.key
pi@viper:~/ORIONSmartDoorBell/certs $ ls -l /etc/mosquitto/certs/
total 16
-rw-r--r-- 1 root root 1424 Jul 11 14:34 orion_ca.crt
-rw-r--r-- 1 root root 130 Sep 30 2023 README
-rw-r--r-- 1 root root 1298 Jul 11 14:34 ring_server.crt
-rw----- 1 root root 1675 Jul 11 14:34 ring_server.key
```

- The solution is to change the ownership of `ring_server.key` to mosquito user account with following command: `sudo chown mosquito:mosquitto ring_server.key`
 - Note: Be certain to ensure that ONLY the mosquito user has read/write privileges. No other user should have access. Otherwise, your TLS configuration is comprised because the private key leaked*

Broker service Restarted Successfully...

Proper file permissions:

- private key is secret (must be unreadable by all except the owner user/process) “mosquitto” user, and super user: “root”
- The certificate files are public
 - CA cert loaded in browser trusted store
 - Server cert is handed over to the client app (browser), to prove the server’s identity?

```
pi@viper:/etc/mosquitto/certs $ ls -l
total 16
-rw-r--r-- 1 root      root      1424 Jul 11 14:34 orion_ca.crt
-rw-r--r-- 1 root      root      130  Sep 30  2023 README
-rw-r--r-- 1 root      root      1298 Jul 11 14:34 ring_server.crt
-rw----- 1 mosquitto mosquitto 1675 Jul 11 14:34 ring_server.key
pi@viper:/etc/mosquitto/certs $ sudo systemctl restart mosquitto
pi@viper:/etc/mosquitto/certs $
```

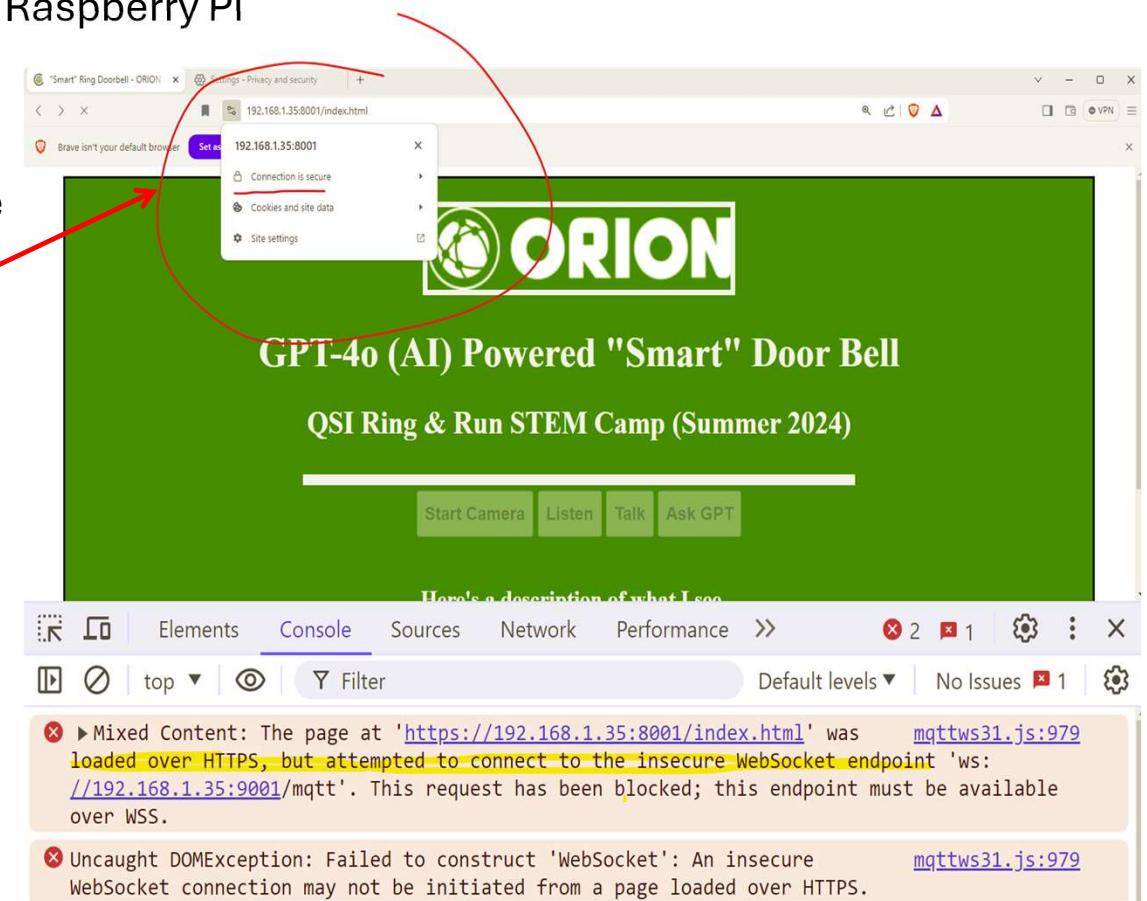


If your certificate is stolen, and your common name (www.domain.com) spoofed, would your identity be comprised?
(group discussion to follow)

View the Doorbell App (Secure mode) With PKI Certs. Installed, and Broker Configuration Updated

Open Brave and enter following URL into the address bar: [https://\[IP-address\]:8001/index.html](https://[IP-address]:8001/index.html)
Replace [IP-address], with IP address of your Raspberry Pi

- We updated the Broker configuration,
BUT... the same error is displayed as before
 - Notice the error message about an **insecure WebSocket using endpoint:**
'ws://[ip-address]:9001/mqtt'
 - Refer to updated Broker configuration file,
and **see the next slide for a hint**



Update the Client Application (JavaScript) Code to Use TLS

- In the VSCode editor, open the JavaScript file located in: [wwwroot/js/client_app.js](#)
 - Recall that the SSL/TLS enabled “listener” we specified for the MQTT broker configuration uses web sockets exposed on TCP/TLS port 8883
 - We need to update the [client_app.js](#) code to use this SSL/TLS configuration
 - **Challenge Question:** Review the [client_app.js](#) code and identify the changes required to make code compliant with above mentioned TLS configuration. (group discussion to follow)
 - Hint: there are two different lines of code that need to be updated



The screenshot shows the Visual Studio Code interface with the title bar "ORIONSmartDoorBell [SSH: 192.168.1.35]". The Explorer sidebar on the left shows a file tree for "ORIONSMARTDOORBELL [SSH: 192.168.1.35]" with folders like ".pycache_", ".vscode", "certs", "sounds", "wwwroot" (containing "css", "html_pages", "client_ring_app.html", "images", and "js"), and files "client_app.js" (marked with a "M" indicating it's modified), "audioUtils.py", and "ring_server.py". The main editor tab is "client_app.js", which contains the following code:

```
const camera_image = document.getElementById('camera_image');

const messageDiv = document.getElementById('response');
const camera_button = document.getElementById('camera_control');
const gpt_button = document.getElementById('gpt_control');
const listen_button = document.getElementById('listen_control');
const talk_button = document.getElementById('talk_control');
const audio_player = document.getElementById("audioPlayer");

const REMOTE_APP_CAMERA_ONOFF_CONTROL_TOPIC = "ring/remote_app_control/camera"
const REMOTE_DEV_CAMERA_ONOFF_CONTROL_TOPIC = "ring/local_dev_control/camera"
const REMOTE_APP_MICROPHONE_CONTROL_TOPIC = "ring/remote_app_control/microphone"
const REMOTE_APP_AUDIO_DATA_TOPIC = "ring/remote_app_audio_data"

const GPT_RESPONSE_TOPIC = "ring/gptresponse"
const GPT_REQUEST_TOPIC = "ring/gptrequest"
const LISTEN_AUDIO_RESPONSE_TOPIC = "ring/audioresponse"

let BROKER_PORT = 9001
let is_connected = false
let mediaRecorder;
let audioChunks = [];

async function SetupMediaRecorder() {
    try {
        const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
        mediaRecorder = new MediaRecorder(stream);

        mediaRecorder.ondataavailable = (event) => {
            if (event.data.size > 0) {
                audioChunks.push(event.data);
            }
        };
        mediaRecorder.onstop = () => {
```

Review (back-end) Server for TLS

- In the VSCode editor, open the server code file: *ring_server.py*
 - **Challenge Questions:**
 1. Study the code on lines 330 - 350 and describe what these accomplish?
 2. Notice line 301: “*host=“127.0.0.1”*”
 - Describe what “127.0.0.1” is, and how it is being used in the Doorbell server ?



Finally, View the App in Secure Mode Again.. It Works!! Congratulations!

You have successfully implemented TLS into the Doorbell Design

Spend a couple of minutes interacting with the app

Notice "Ask GPT" doesn't function as expected

Exercise: you might try on your mobile device. Email the CA certificate (only) to yourself, and from your mobile device, open and click on certificate to install it.

