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

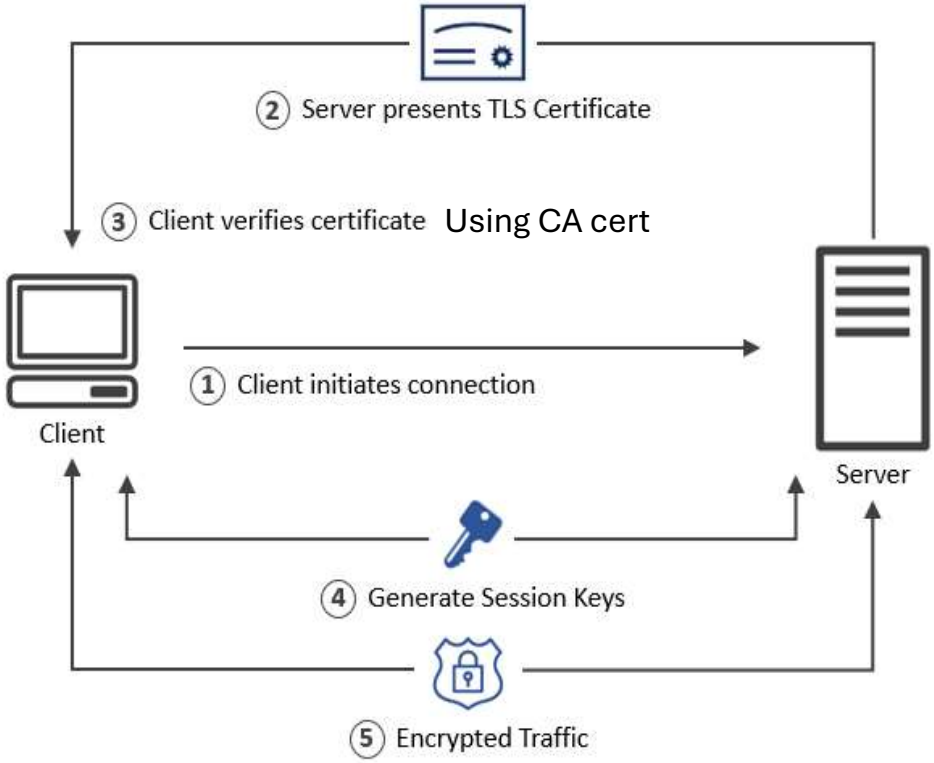
- What is SSL/TLS (HTTPS)?
  - https://www.youtube.com/watch?v=j9QmMEWmcfo

TLS (Concept) Illustrated

Example Usage:

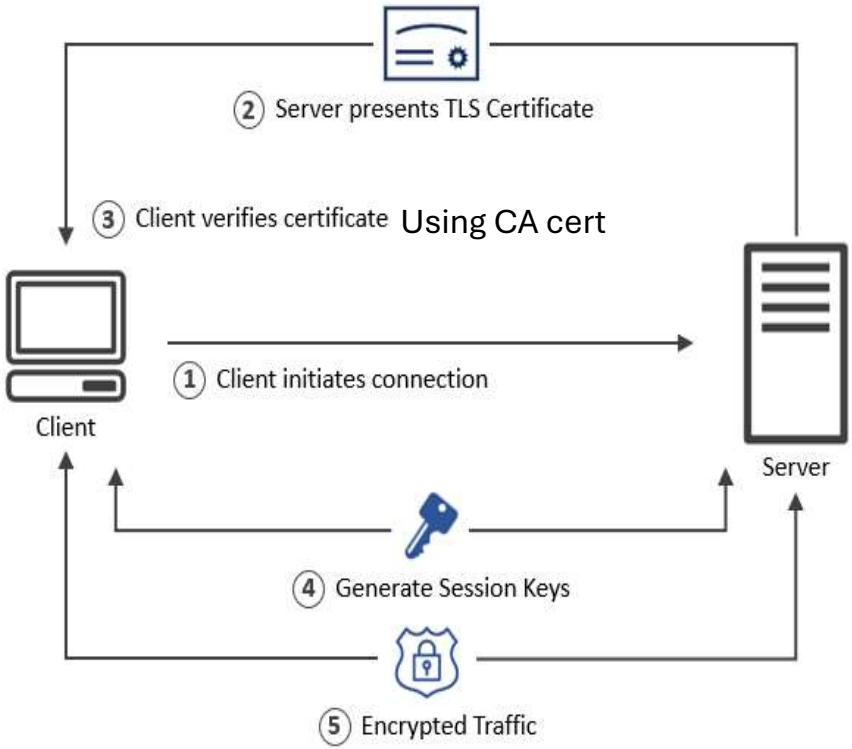
- A typical web browser and web server connection using **SSL/TLS** (**HTTPS**):
  - This connection is used on the Internet to send email in Gmail, and when doing online banking, shopping etc.
    1. Browser connects to server using https.
    2. Server presents certificate to the client (Browser)
    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/

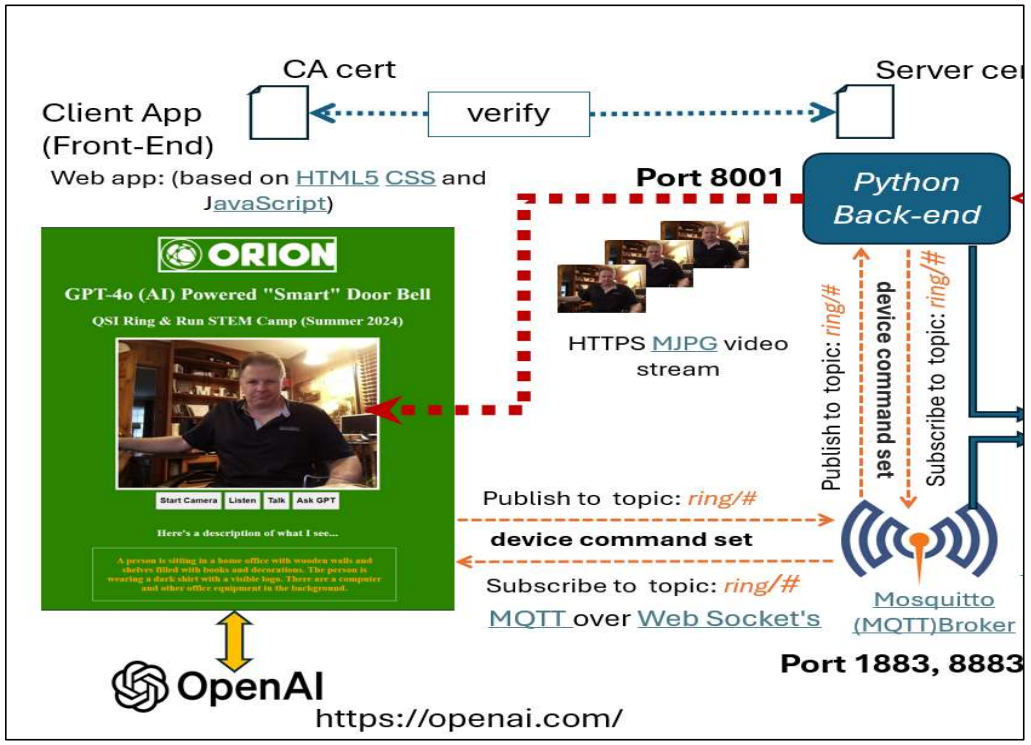


Source: https://www.csa.gov.sg/Tips-Resource/internet-hygiene-portal/information-resources/tls

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_certifying_authorities

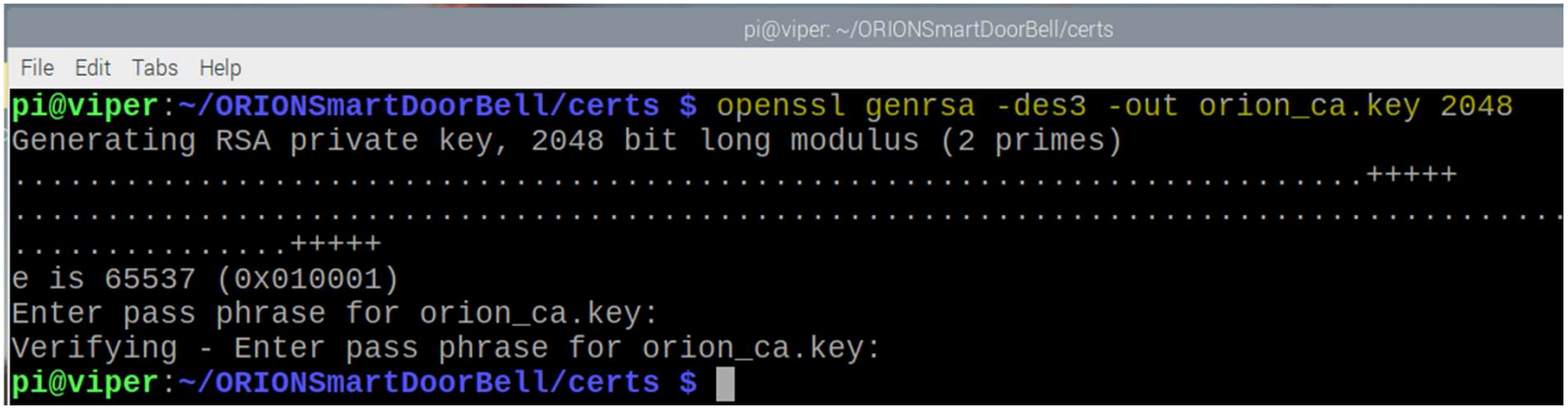
Or

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”.

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



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 it's 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 it's public key
        - The CA certificate in (the CA chain containing the CA's public key) is publicly available, and used to very 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)

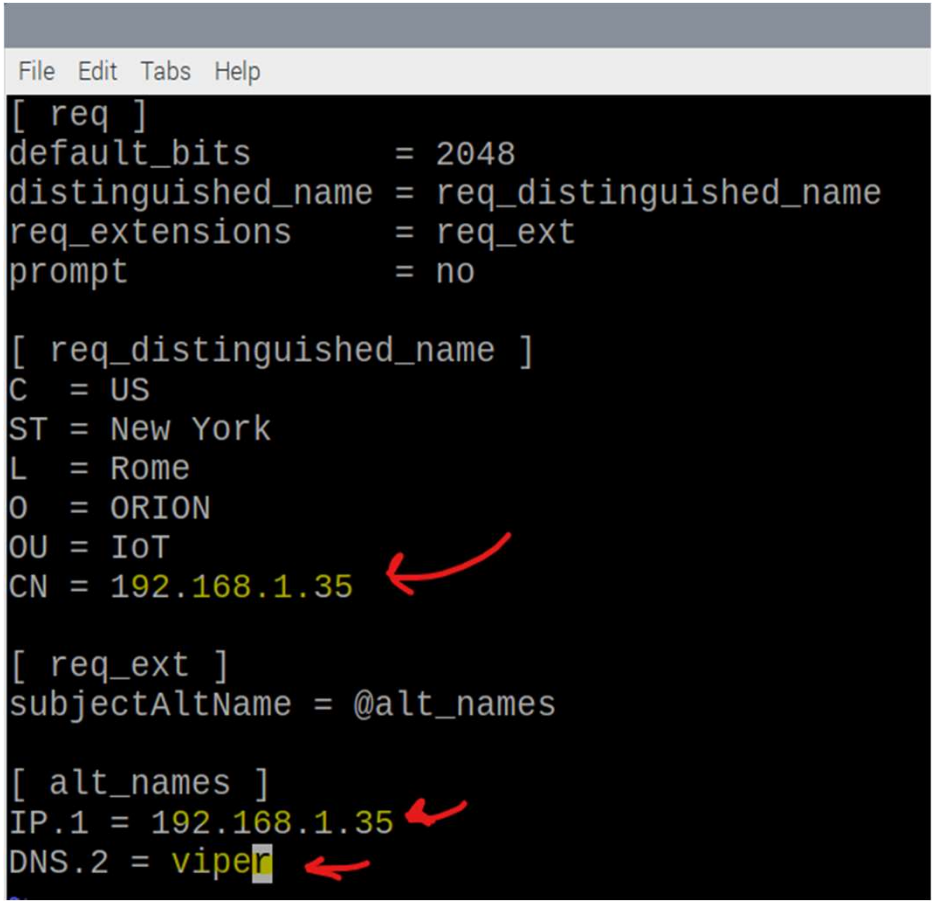
Step 4 of 5: Create Server Certificate Request (CSR) cont.

- 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



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 requestioning 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

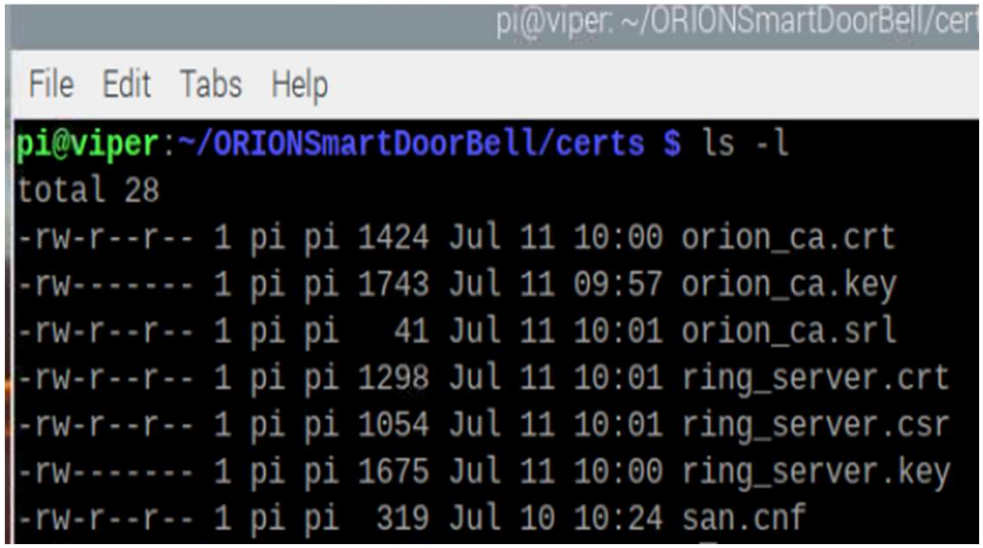
```
pi@viper:~/ORIONSmartDoorBell/certs $ openssl x509 -req -in ring_server.csr -CA orion_ca.crt -
CAkey orion_ca.key -CAcreateserial -out ring_server.crt -days 365 -extensions req_ext -extfile
 san.cnf
Signature ok
subject=C = US, ST = New York, L = Rome, O = ORION, OU = IoT, CN = 192.168.1.35
Getting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



- 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
    - *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

Client App (Front-End)

Web app: (based on HTML5 CSS and JavaScript)

OpenAI

https://openai.com/

HTTPS MJPG video stream

Python Back-end

Publish to topic: ring/#

device command set

Subscribe to topic: ring/#

Publish to topic: ring/#

device command set

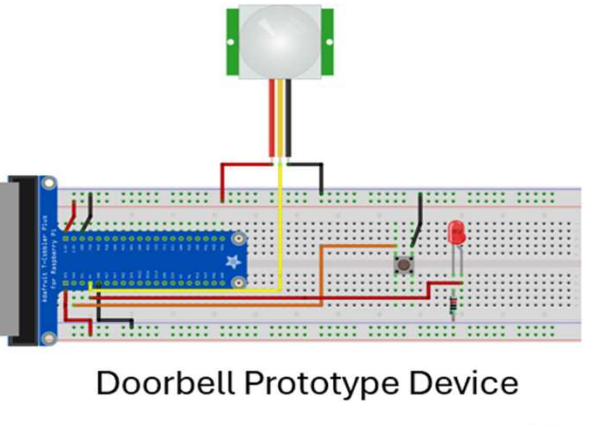
Subscribe to topic: ring/#

MQTT over WebSockets

Mosquitto (MQTT)Broker

Raspberry Pi 4B

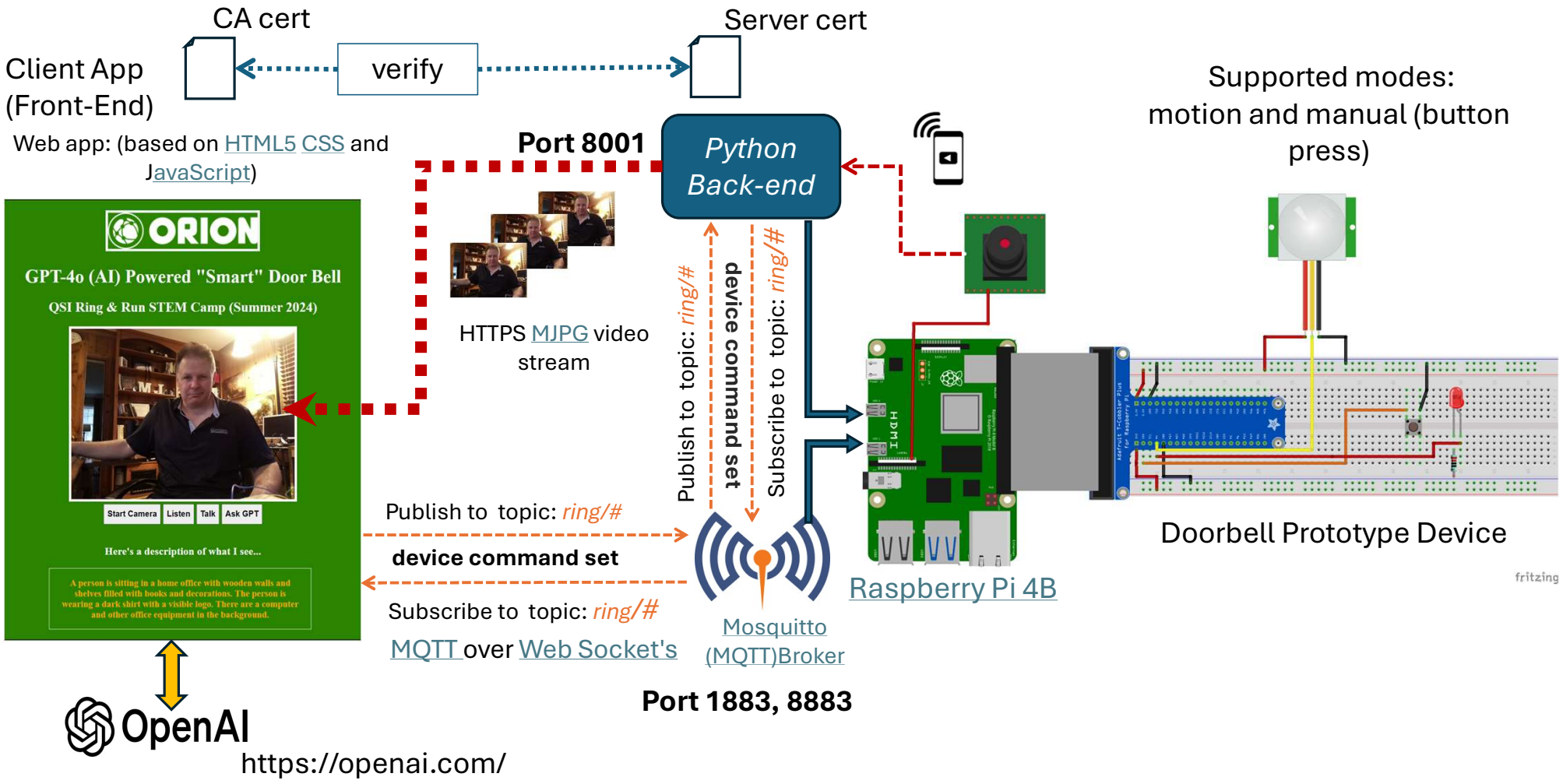
Supported modes: motion and manual (button press)



Doorbell Prototype Device

fritzing

IoT enabled “Smart” Doorbell Concept with TLS

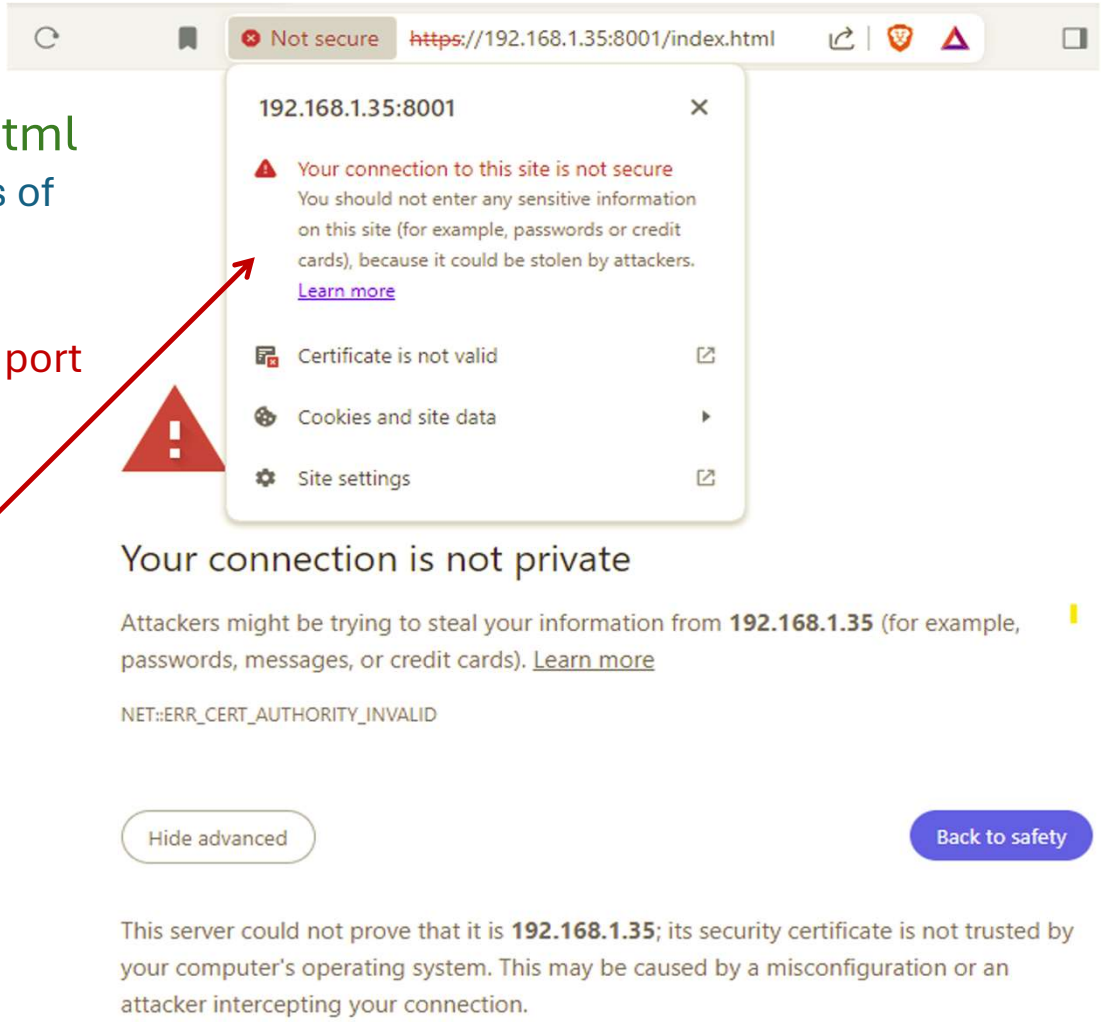


Viewing the Doorbell Application (Secure mode)

- 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?



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.

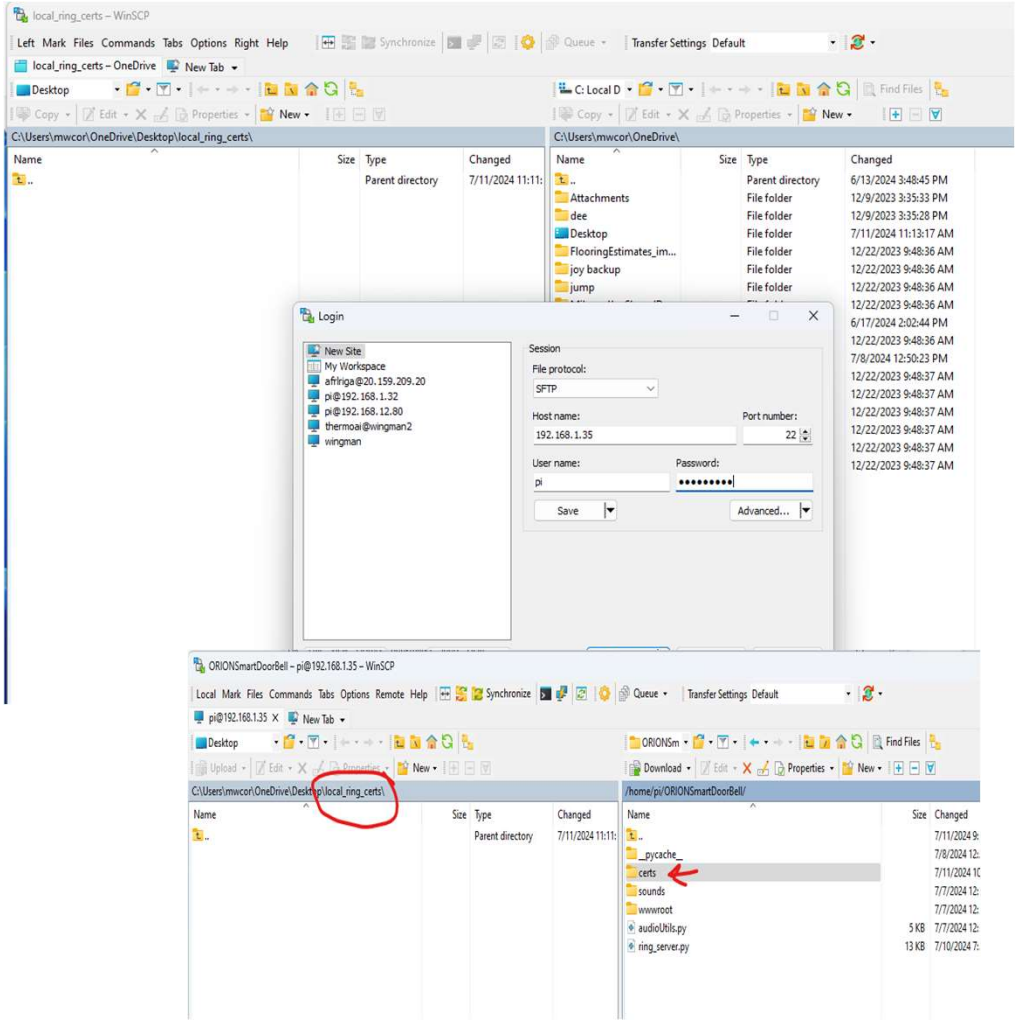
- To accomplish this, we begin by adding the ORION CA certificate to the browser's "Trusted Store" - a folder on the device where certificates are stored/managed
  - We will accomplish this with "Brave "browser that we previously installed on the laptop.

Use WinSCP to copy the Certs Folder

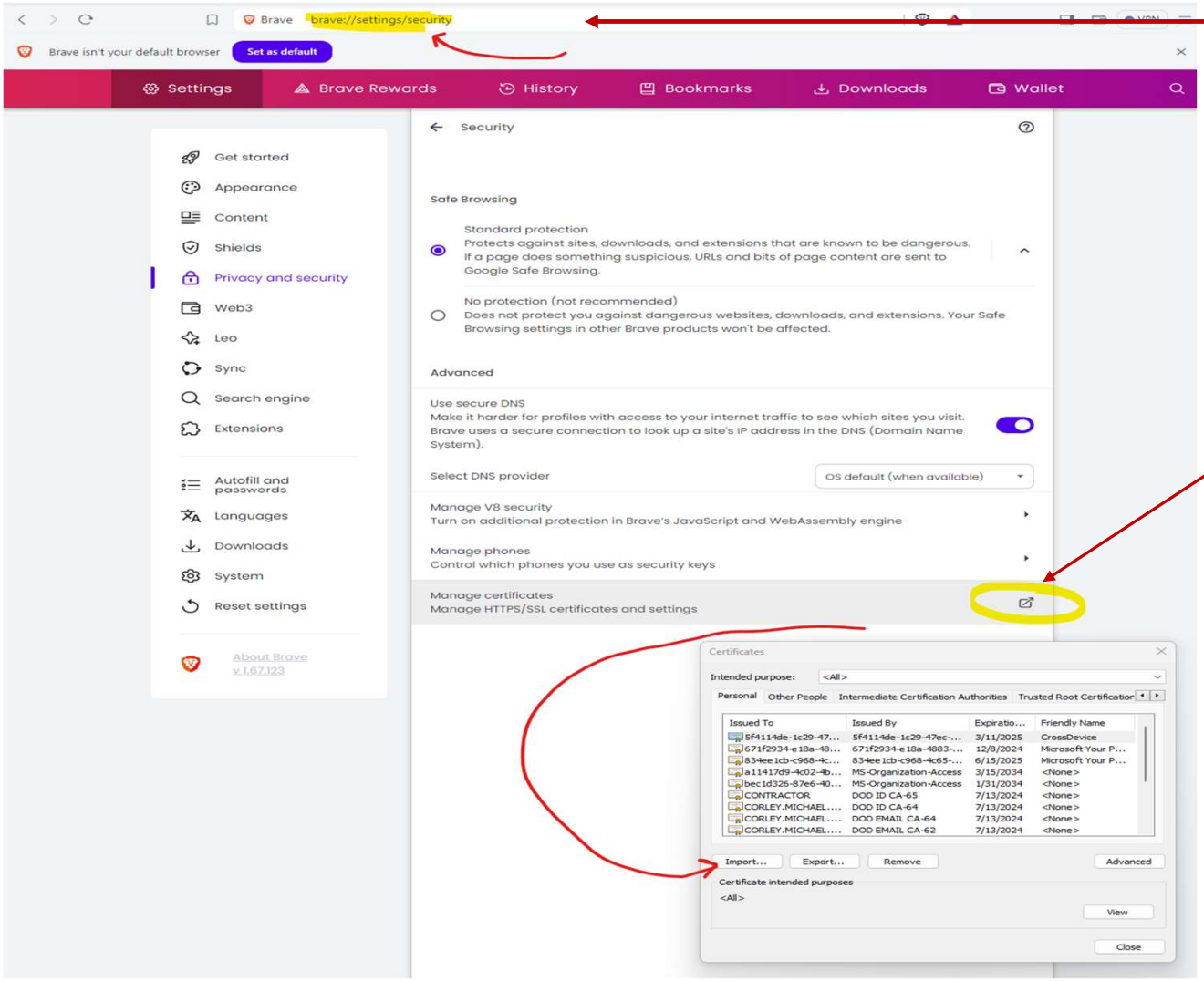
- We need the certificates (currently stored on the Raspberry Pi) copied to the Windows laptop so we can import the ORION CA cert into the Brave browser's "Trusted Store" location:
  - For this we will use WinSCP
    - Download WinSCP here: https://winscp.net/eng/download.php

1. Create a folder on the Windows 11 Desktop, named: "local_ring_certs"

2. 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.)

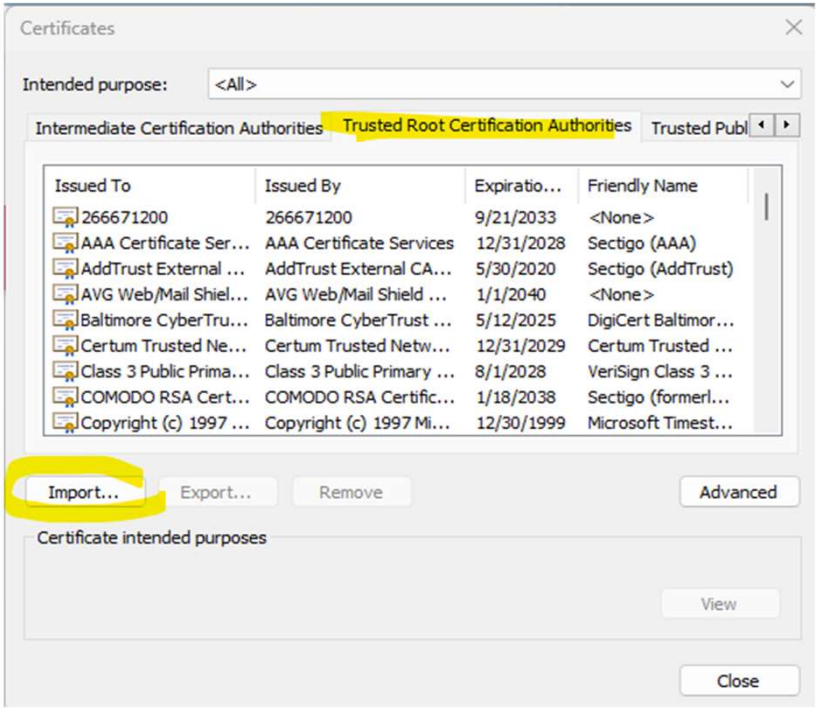


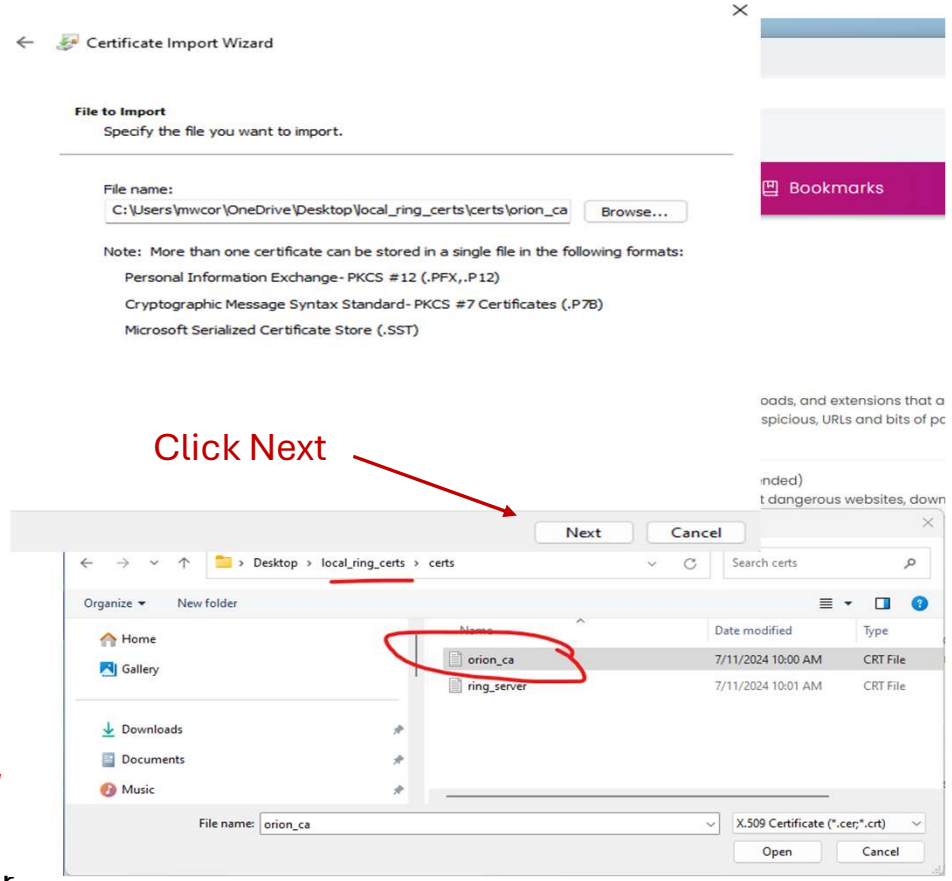
1. Open Brave browser and enter the following in the address bar: *brave://settings/security*

2. *Click "Manage Certificates"*

3. See next slide

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

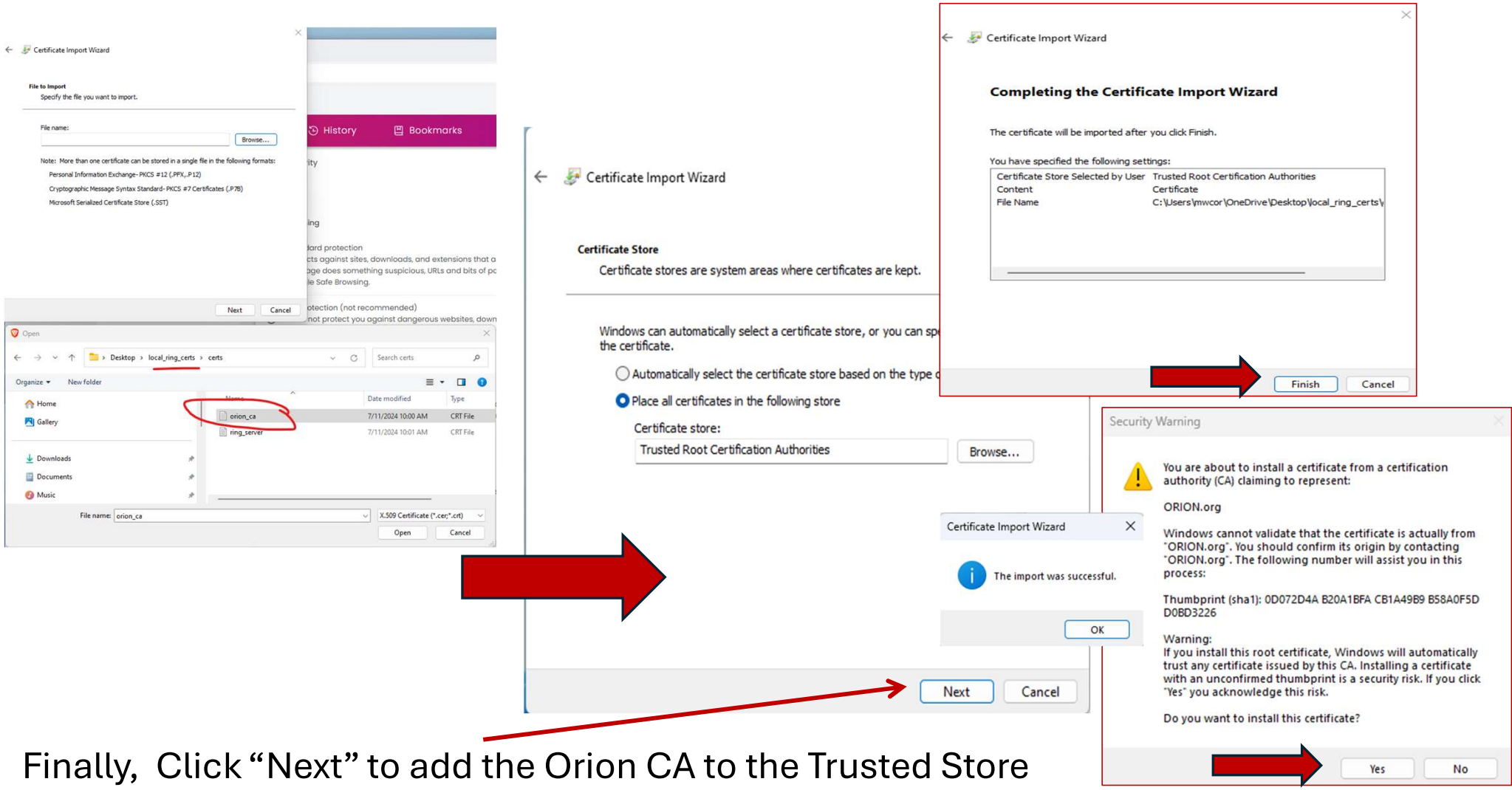




1. *Click "Trusted Root Certifications" Tab*
   - *Notice the official list of Trusted CA certs displayed*

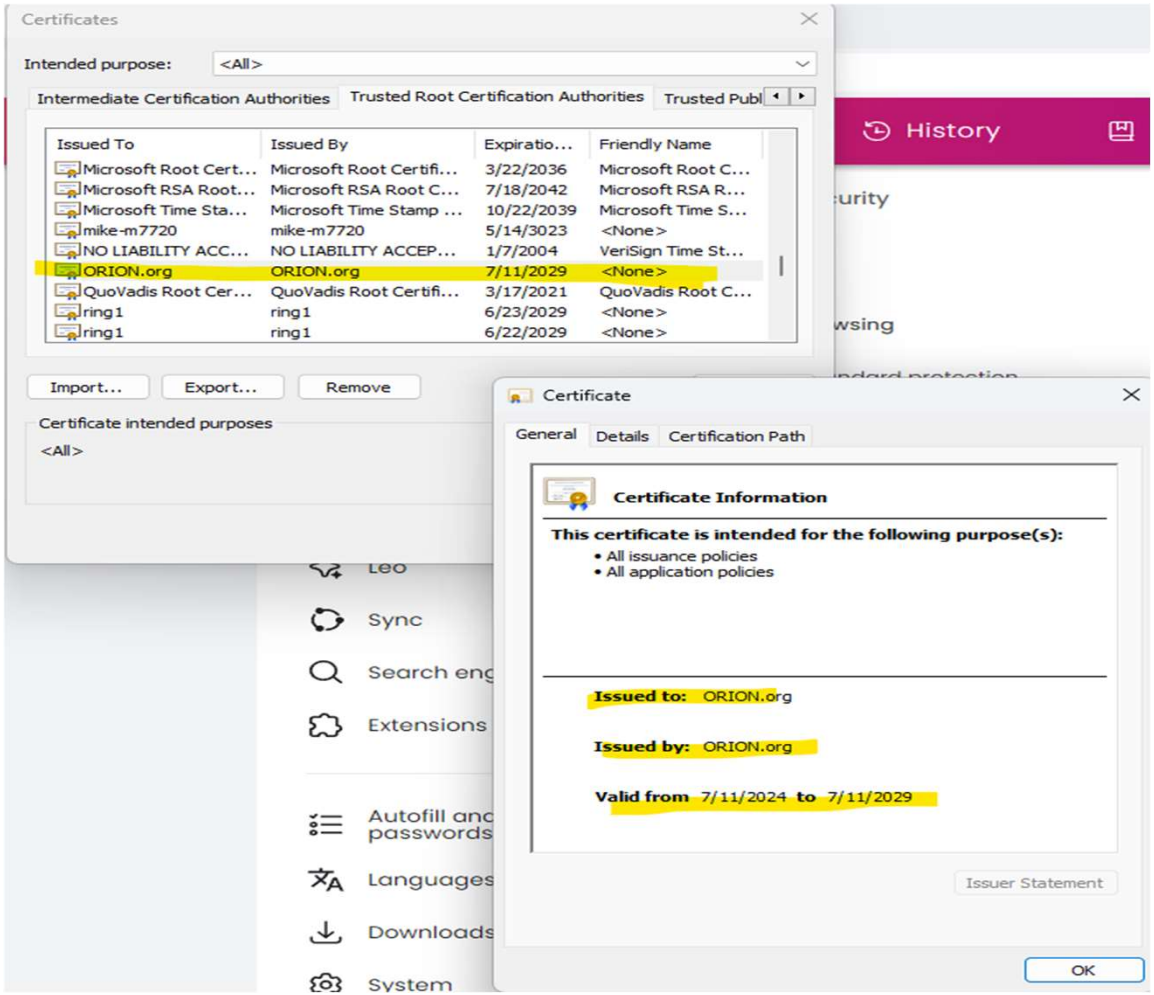
2. Click "Import" and navigate to "local_ring_certs" folder on the Windows Desktop, select "orion_ca.crt" and click "Open"

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



Finally, Click “Next” to add the Orion CA to the Trusted Store

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
    - *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
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.

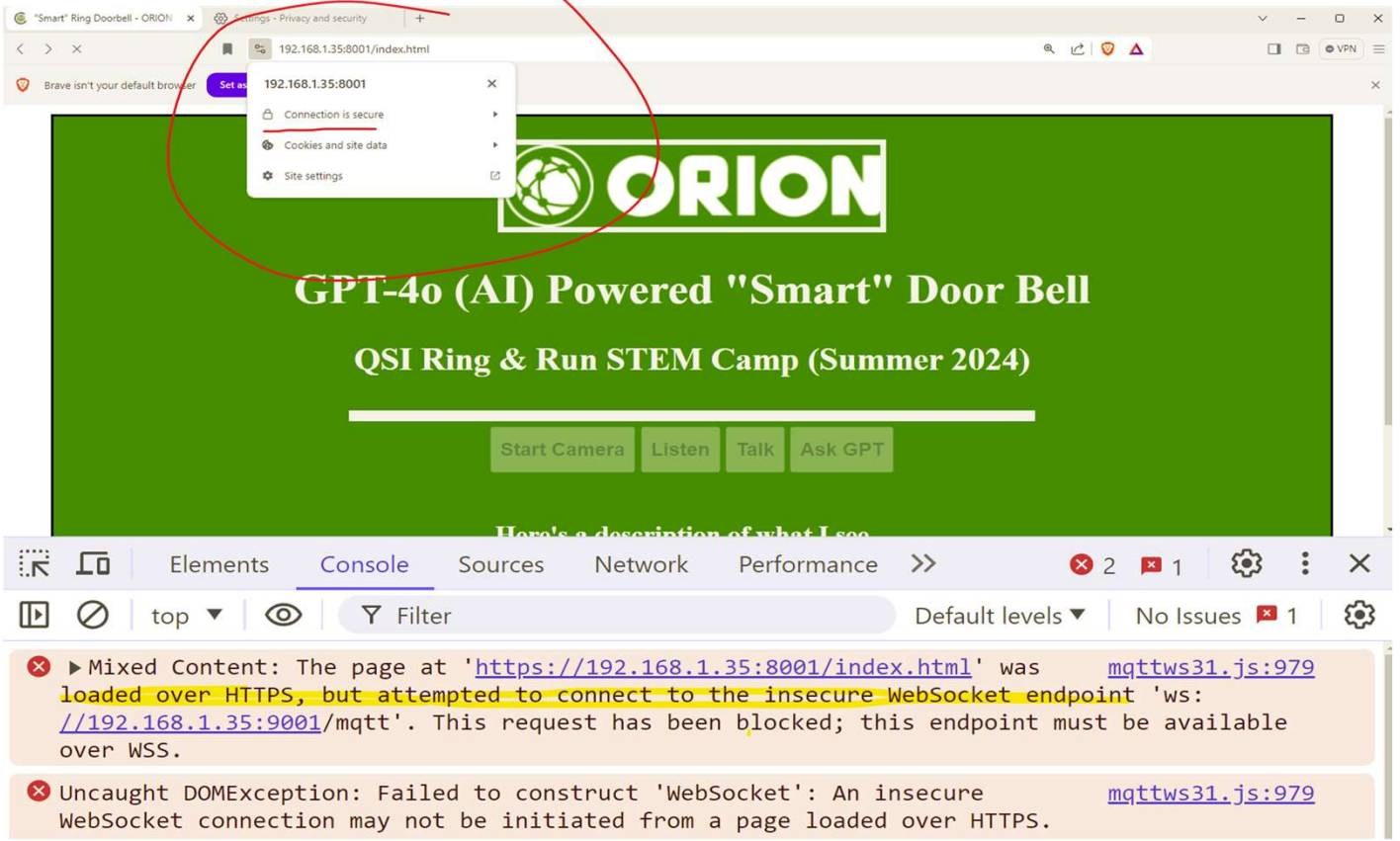
Challenge Question: What does this error message indicate ?

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

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!



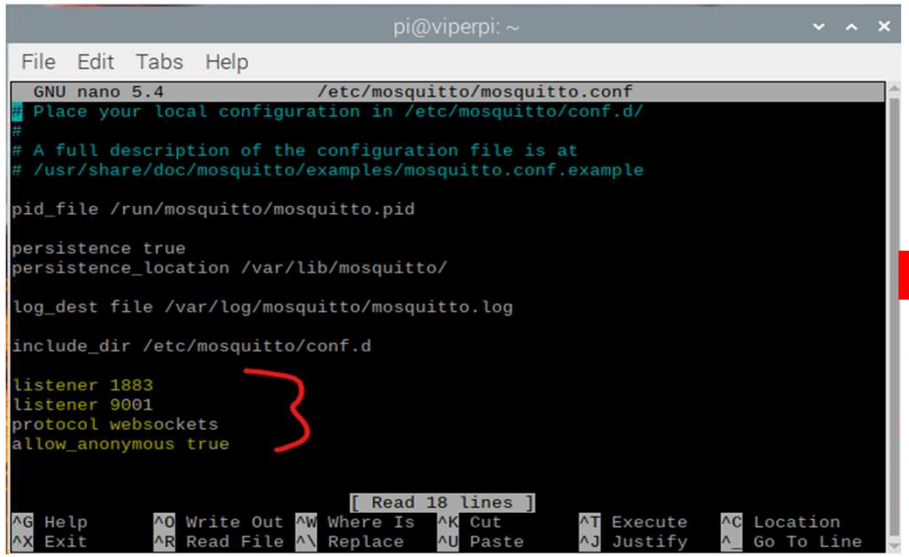
Update the MQTT Broker Configuration (see Slide 69 for the current configuration)

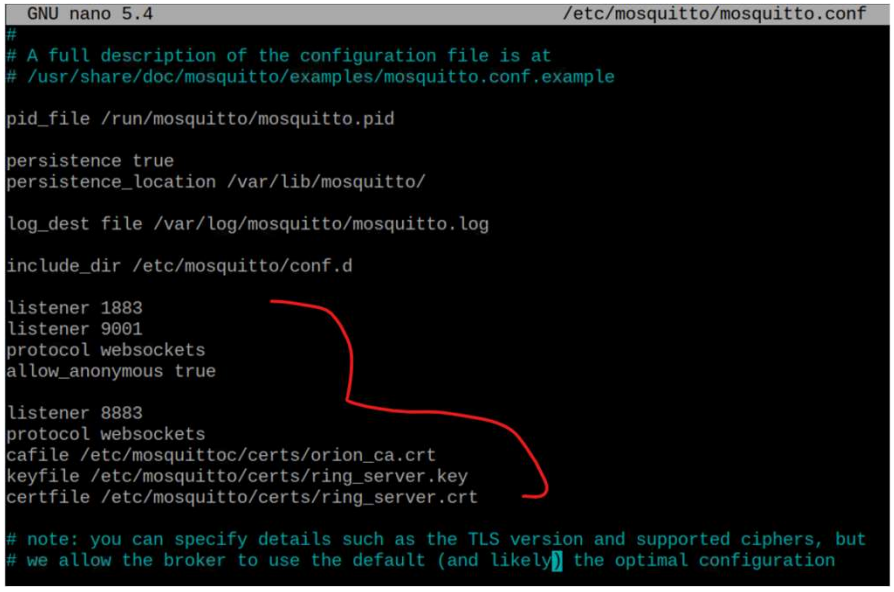
- From VNC session, or from the VSCode terminal, and open the mosquito broker configuration file: *sudo nano /etc/mosquitto/mosquitto.conf*
  - The curent configuration file should resemble the screen shot below.
- To update the Broker configuration for compliance with TLS, we need to add another “listener” 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:





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: *<u>/etc/mosquitto/certs</u>***

```
  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

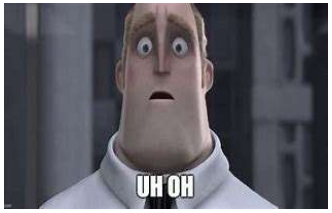
- From VNC session terminal (or from the VSCode terminal), navigate to the doorbell project “certs” subfolder and copy certificate files to Broker configuration folder with the following command:
  sudo cp orion_ca.crt ring_server.key ring_server.crt /etc/mosquitto/certs/

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 mosquitto:mosquitto ring_server.key*
  - *Note: Be certain to ensure that ONLY the mosquitto 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

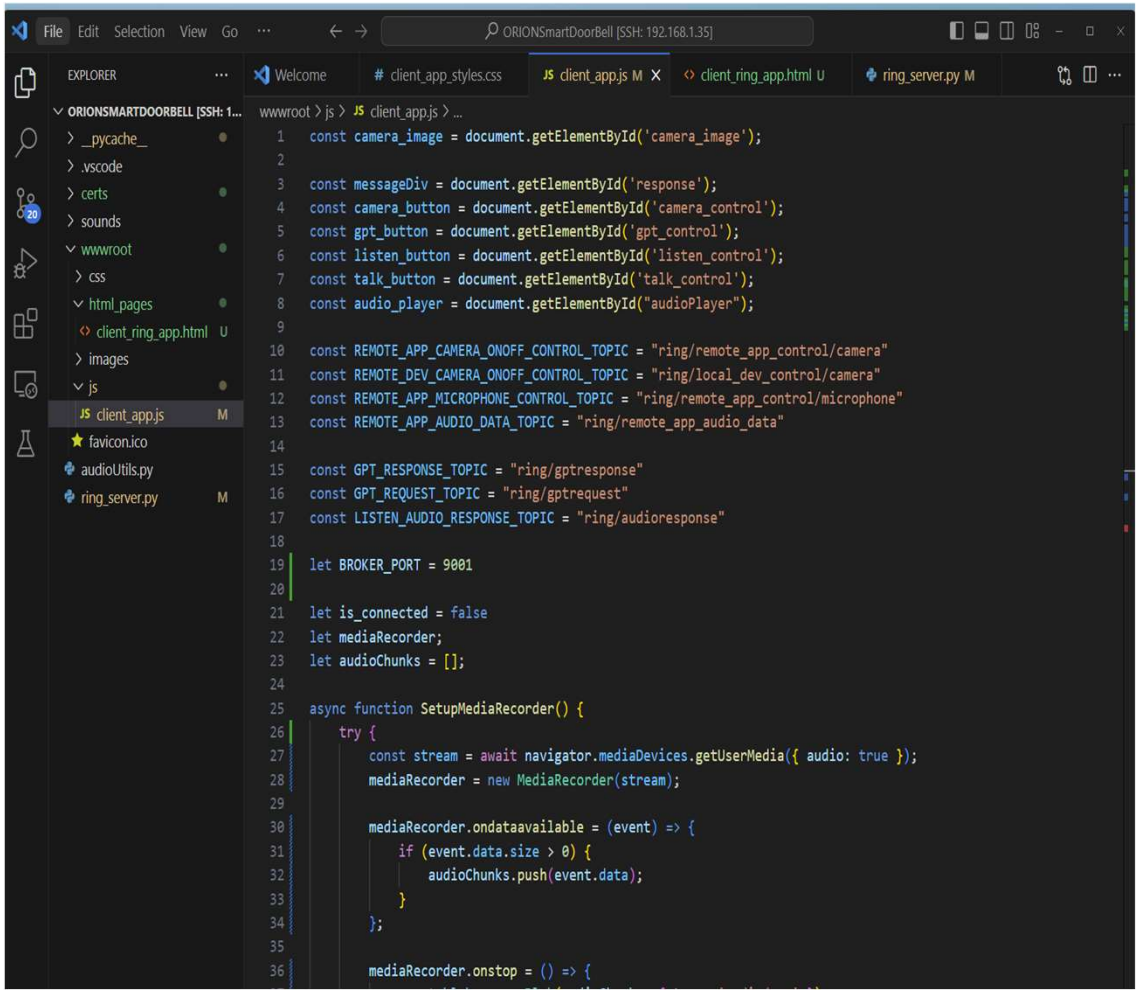
Open Brave and enter following URL into the address bar: https://[IP-address]:8001/index.html
Replace [IP-address], with IP address of your Raspberry Pi

- 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 identity 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



Review (back-end) Server for TLS

- In the VSCode editor, open the server code file: *ring_server.py*
- **<u>Challenge Questions</u>**:
  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 ?

ring_server.py M | client_app.js M

ring_server.py > ...

```
305  host="127.0.0.1"
306  doorbell_sound_file_path = "./sounds/bell1.mp3"
307
308  BUTTON_GPIO_PIN=2
309  MOTION_SENSOR_GPIO_PIN=4
310
311  client = paho.Client(paho.CallbackAPIVersion.VERSION2, transport="tcp")
312  client.on_message = on_message;
313  client.on_connect = on_connect
314
315  client.on_disconnect = on_disconnect
316
317  # Initialize pygame mixer
318  pygame.mixer.init()
319
320  button = Button(BUTTON_GPIO_PIN)
321  pir = MotionSensor(MOTION_SENSOR_GPIO_PIN)
322
323  if args.mode == "motion":
324      pir.when_motion = handleMotionMode
325
326  button.when_pressed =  handleButtonMode
327
328  with picamera.PiCamera(resolution='1024x768', framerate=24) as camera:
329      output = StreamingOutput()
330      # camera.rotation = 180
331
332      try:
333          ap = AudioPlayback()
334          ap.SetMQTTClient(client, LISTEN_AUDIO_RESPONSE_TOPIC)
335
336          # streaming video (web server) address and port
337          address = ('', HTTP_SERVER_PORT)
338          if args.secure == "on":
339              address = ('', HTTPS_SERVER_PORT)
340
341          server = StreamingServer(address, StreamingHandler)
342
343          print("\"Smart\" Doorbell server started on port: " + str(address[1]))
344          if args.secure == "on":
345              # 1. configure the Python HTTP server for HTTPS (TLS support)
346              # wrap the TCP socket with an SSL support, then load the certs.
347              server.socket  = ssl.wrap_socket (server.socket,
348                  keyfile="./certs/ring_server.key",
349                  certfile="./certs/ring_server.crt",
```

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 the with 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.