

# MQTT Security

Frank Walsh

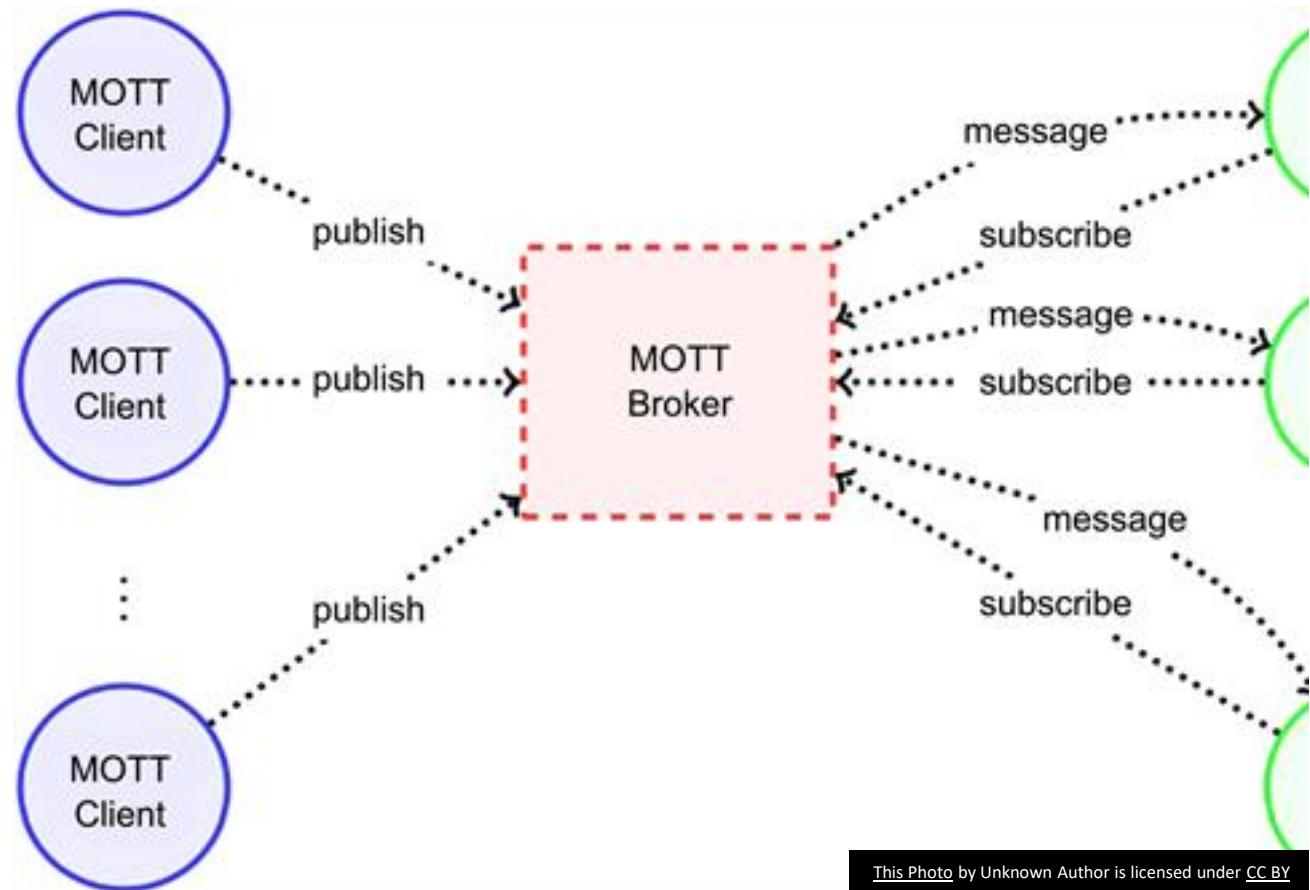


# Security in IoT

- Internet of Things presents new security implementation challenges
  - IoT devices have limited computing power and memory capacity
  - Cryptographic algorithms may require more resources than constrained IoT devices possess
  - Critical security issues that require updates to be rolled out to all devices simultaneously can be affected by unreliable networks on which many IoT devices
- **Security is a critical concern for developers of IoT applications.**

# Security in MQTT

- As with Web Dev, a “layered” approach over the protocol stack
- MQTT specifies only a few security mechanisms
- MQTT uses other accepted security standards at the various levels:
  - E.g. SSL/TLS for transport security.
- Network Layer: Use a VPN for all communication between clients and brokers. Gateway-based applications, the gateway is connected to devices on the one hand and with the broker over VPN on the other side.
- Transport Layer: TLS/SSL
- Application Layer: Authorisation using username/password provided by the Protocol



This Photo by Unknown Author is licensed under CC BY

# MQTT authentication

- The MQTT protocol provides username and password fields in the CONNECT message for authentication
  - Username is an UTF-8 encoded string.
  - Password is binary data with a maximum of 65535 bytes.
- The MQTT broker evaluates credentials and returns one of the following return codes:
  - 0 – Connection Accepted
  - 4 – Connection Refused, bad username/password
  - 5 – Connection Refused, not authorised
- Username/password sent as plain text - **Secure transmission of usernames and passwords requires transport encryption**

# MQTT authentication/authorisation

- Authorization specifies access rights to a resource.
- In addition to the username and password, MQTT clients provide other information that can be used for authentication.
  - Client Identifier (Client ID)
  - X.509 Certificate
- Without proper authorization, each authenticated client can publish and subscribe to all available topics
  - Topic permissions must be implemented on the broker side.
- Permissions are configurable as follows:
  - Allowed topic (exact topic or wild card topic)
  - Allowed operation (publish, subscribe, both)
  - Allowed quality of service level (0, 1, 2, all)



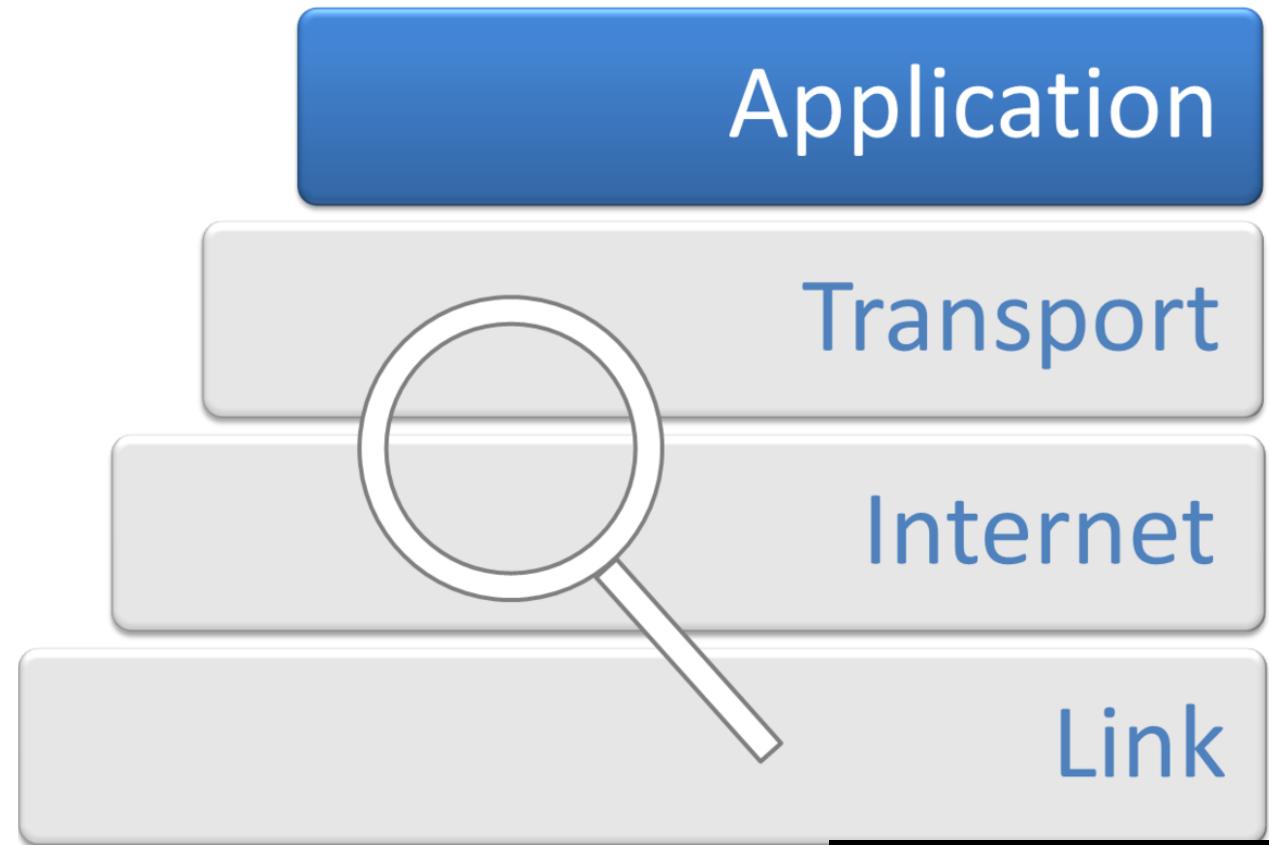
# Transport Layer Security (TLS) and Secure Sockets Layer (SSL)

- TLS and SSL provide a secure communication channel between a client and a server at **Transport Layer**.
- TLS and SSL are cryptographic protocols which use a handshake mechanism to negotiate various parameters to create a secure connection between the client and the server.
- Servers provide a **X509** certificate (typically issued by a trusted authority) that clients use to verify the identity of the server.
- While the additional CPU usage is typically negligible on the broker, small constrained devices are not designed for computation-intensive tasks.
- If TLS is not possible, payload encryption and at least hash or encrypt the password in the CONNECT message of your client.



# Application Level Encryption

- End to End encryption between publisher and subscriber
- Encrypt the payload without having to configure the MQTT broker
  - Means publisher and subscriber applications have to “know” encryption/decryption method. Has to be coded
- Possible alternative for constrained devices that cannot support TLS
- Possible (less secure) alternative for using public brokers.



[This Photo](#) by Unknown Author is licensed under CC BY-SA

# Creating MQTT Clients in Python: Publishing Client

- We'll use **Paho** MQTT python client from Eclipse
- Import the **client class** to provide functions to publish messages and subscribe to topics on MQTT brokers.
- Create instance and connect to a broker

```
import paho.mqtt.client as mqtt #import the client1
broker_address="192.168.1.184"
#broker_address="iot.eclipse.org" #use external broker
client = mqtt.Client("P1") #create new instance
client.connect(broker_address) #connect to broker
client.publish("house/main-light","OFF")#publish
```

# Creating MQTT Clients in Python: Subscribing Client Callback

- When the client receives messages it generate the `on_message` callback.

```
import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print("message received " ,str(message.payload.decode("utf-8")))
    print("message topic=",message.topic)
    print("message qos=",message.qos)

def main():
    broker_address="192.168.1.184"
    client = mqtt.Client("P1") #create new instance
    client.connect(broker_address) #connect to broker
    client.subscribe("house/main-light")

if __name__ == "__main__":
    main()
```