SENQUIP DEVICES AND THIRD-PARTY MQTT ENDPOINTS

1. Introduction

Senquip devices typically send data to the Senquip Portal using secure MQTT. Where third-party integration is required, device data can be retrieved via the Senquip API, or the device can connect directly to a third-party MQTT or HTTP endpoint.

Use of the Senquip API is preferred because Senquip optimises the data path to the Portal, performs server load balancing, and manages device certificates, encryption, and broker security in accordance with our Information Security Management System (ISMS).

If data must be sent directly to a custom endpoint, this can be achieved using MQTT or HTTP. MQTT is generally recommended because HTTP sends data as individual transactions, opening and closing a connection each time, whereas MQTT maintains a persistent connection to a broker and exchanges small messages continuously.

It is important to recognise that secure MQTT deployments — particularly those using mutual TLS (mTLS) — require certificate provisioning, trust management, and ongoing lifecycle control for each device. When connecting directly to third-party endpoints, these responsibilities shift to the endpoint owner and must be managed securely and at scale.

This document describes how to connect a Senquip device to a third-party MQTT broker and verify publish and subscribe operation using the Mosquitto command-line tools and a public test broker. It also outlines the security considerations that arise when implementing production-grade secure MQTT connections.

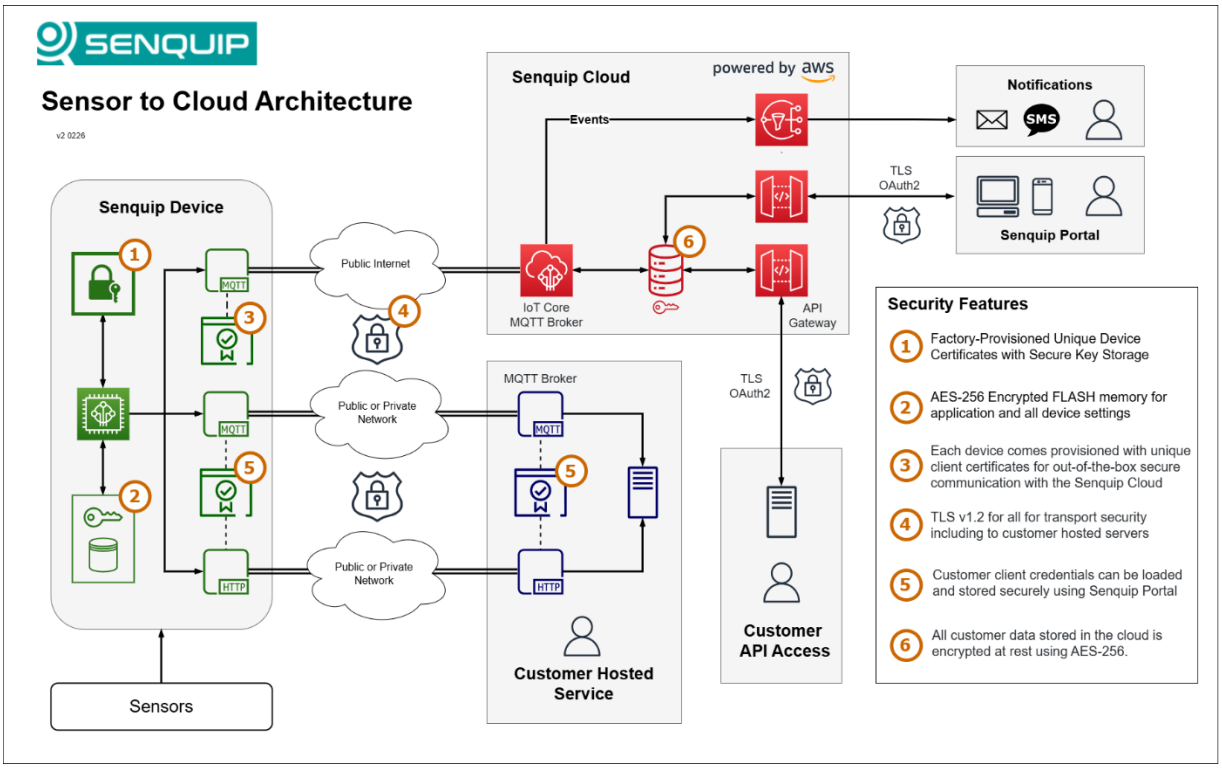


Figure 1 - Senquip Cloud Architecture

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

2. Introduction to MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for unreliable networks such as cellular or satellite connections. Senquip uses MQTT as the protocol for connection to the Senquip Portal.

MQTT is commonly used in telemetry because it:

- **Very low data usage** – tiny headers and persistent connection ideal for cellular devices
- **Reliable on poor networks** – built-in retries and delivery guarantees (Quality of Service, QoS)
- **Near real-time communication** – server can instantly send commands to devices (no polling)
- **Decoupled architecture** – devices talk to a broker, so back-end systems can change without affecting devices
- **Low power consumption** – fewer reconnects and less radio time means less power consumed

2.1. Broker

Core to the MQTT protocol is the broker Devices publish messages to the broker, and the broker forwards those messages to subscribed clients. This allows multiple systems to exchange data without needing direct connections to each other. In the case of Senquip, devices typically publish data messages and subscribe to receive settings updates.

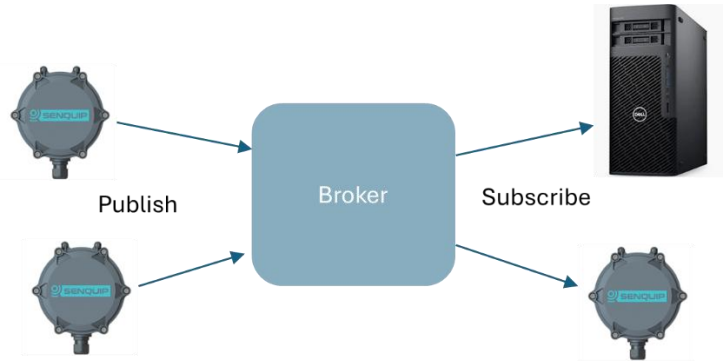


Figure 2 - MQTT Publish and Subscribe Architecture

Example Public test brokers

- test.mosquitto.org
- broker.hivemq.com
- broker.emqx.io

Public test brokers are intended for connectivity testing only. They are shared, publicly accessible services and should not be used for production deployments. Authentication controls may be limited, and data confidentiality cannot be guaranteed. Do not send sensitive or production data to public test brokers.

2.2. Topics

When multiple devices publish messages to an MQTT broker, topics provide logical segregation of those message streams. A device publishes data to a specific topic, and clients subscribe to the topics they require.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

This publish/subscribe model separates devices from clients, allowing many devices to communicate with many subscribers without being directly connected to each other.

Topics are structured like paths, using / as a separator. Typical topics for Senquip devices may be:

- senquip/<deviceID>/telemetry
- senquip/<deviceID>/alarms
- senquip/<deviceID>/control

Structuring topics in this way allows a client to subscribe only to the information it requires. For example, a client could subscribe to: *senquip/deviceID/alarms* to only receive alarm messages for a particular device.

2.3. Client ID

Each MQTT client must present a unique Client ID when connecting to a broker.

The Client ID:

- Identifies the session to the broker
- Enables session state and message queuing
- Prevents duplicate connections

If two clients connect with the same Client ID, the broker will disconnect one of them.

For Senquip devices, the default Client ID is the Device ID and should remain unchanged unless specifically required by the target platform.

2.4. Quality of Service

MQTT defines three Quality of Service levels that determine message delivery guarantees between client and broker:

QoS	Description	Behaviour
0	At most once	Message sent once, no retry. Lowest overhead.
1	At least once	Message resent until acknowledged. Possible duplicates.
2	Exactly once	Guaranteed single delivery using handshake. Highest overhead.

For most telemetry applications, QoS 0 or QoS 1 is appropriate. QoS 2 is rarely required in IoT deployments due to increased protocol overhead. Senquip uses QoS 1 for communication with the Senquip Portal.

2.5. Security

MQTT is a lightweight messaging protocol and does not provide security by default; security must be explicitly configured.

In production environments, security controls are required to protect data, verify identities, and restrict access. Without these controls, devices may be vulnerable to interception, impersonation, or unauthorised access.

Secure MQTT deployments rely on four core controls:

1. **Encryption** – Protecting data in transit

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

2. **Server identity verification** – Confirming the MQTT broker is genuine
3. **Client authentication** – Verifying the identity of the connecting device
4. **Authorisation** – Controlling what the client is permitted to publish or subscribe to

Each control addresses a different security risk. All four are required for a production-grade deployment.

2.5.1. Encryption

Encryption protects MQTT data as it travels between the device and the broker.

In MQTT systems this is provided by **TLS (Transport Layer Security)**, commonly referred to as *MQTTS*. TLS provides:

- **Confidentiality** – Data cannot be read in transit
- **Integrity** – Data cannot be modified without detection

By convention:

- TLS-enabled MQTT uses port 8883
- Unencrypted MQTT uses port 1883

All MQTT communications with the Senquip Portal use TLS.

2.5.2. Server Identity Verification

TLS not only encrypts the connection — it also verifies the identity of the broker.

During the TLS handshake, the client validates the broker's server certificate against a trusted Certificate Authority (CA).

A **Certificate Authority (CA)** is an organisation that issues and digitally signs certificates so devices and systems can verify identities.

This process ensures the device is connecting to the legitimate server and not an impersonator.

Where a CA certificate is required to validate the broker's certificate, the client (in this instance, the Senquip device) must provide:

- CA certificate (ca.crt)

Unlike desktop operating systems, embedded IoT devices such as Senquip units do not include a built-in public trust store. The required CA certificate must therefore be explicitly uploaded onto the Senquip device.

2.5.3. Client Authentication

Encryption and server verification do not, by themselves, control which devices are permitted to connect.

In many systems, the broker also requires authentication of the connecting client.

Common authentication methods include:

- Username and password
- TLS client certificates (mutual TLS)
- Access tokens or API keys
- Anonymous access (typically for testing only)

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

TLS client certificates are typically **X.509 certificates**, the standard format used in public key infrastructure (PKI) systems.

In production IoT deployments, client certificates are typically unique per device. This enables:

- Individual device authentication
- Certificate revocation
- Per-device access control

Authentication determines whether a client is permitted to establish a session.

When mutual TLS is used, the client must provide:

- Client certificate (client.crt)
- Private key (client.key)

The client certificate contains the device identity and public key. The private key is used during the TLS handshake to prove the device owns that identity. The broker validates the client certificate before allowing the connection.

Mutual TLS is commonly used in industrial IoT platforms such as AWS IoT Core and Azure IoT Hub.

2.5.4. Authorisation

After successful authentication, the broker enforces authorisation rules.

Authorisation controls:

- Which topics the client may publish to
- Which topics the client may subscribe to

Authorisation ensures devices can access only the data required for their intended function.

Authentication answers the question: “Who are you?”

Authorisation answers the question: “What are you allowed to do?”

2.6. Testing for Firewalls

Corporate networks and firewalls often restrict outbound traffic on common MQTT ports (1883 and 8883), which may prevent devices from connecting to a remote broker.

Before troubleshooting application-level issues, it is advisable to confirm that the required TCP ports are reachable from the network.

On Windows systems, this can be verified using the **Test-NetConnection** PowerShell command, which attempts to establish a TCP connection to a specified host and port.

Procedure:

1. Connect a PC to the same network as the Senquip device.
2. Open Windows PowerShell.
3. Execute the following command: Test-NetConnection test.mosquitto.org -Port 1883

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

4. If TcpTestSucceeded : True is returned, the port is reachable.
   If False, the connection is being blocked or refused.

```
PS C:\> Test-NetConnection test.mosquitto.org -Port 1883


ComputerName     : test.mosquitto.org
RemoteAddress    : 2001:41d0:303:4831::1
RemotePort       : 1883
InterfaceAlias   : WiFi
SourceAddress    : 2406:2d40:4d79:3810:39f4:a6ca:10ac:f611
TcpTestSucceeded : True
```

Figure 3 - Successful Port 1883 Test

Repeat the test for TLS (port 8883):

```
PS C:\> Test-NetConnection test.mosquitto.org -Port 8883
WARNING: TCP connect to (2001:41d0:303:4831::1 : 8883) failed


ComputerName     : test.mosquitto.org
RemoteAddress    : 54.36.178.49
RemotePort       : 8883
InterfaceAlias   : WiFi
SourceAddress    : 192.168.1.146
TcpTestSucceeded : True
```

Figure 4 - Successful Port 8883 Test

Note that in Figure 4 we get a warning that IPv6 connectivity fails but that we did succeed using IPv4.

3. Setting up a Non-Secure MQTT Session

This section demonstrates basic MQTT publish and subscribe operation using a public test broker. This is intended for connectivity verification only and must not be used for production data.

To perform these tests, we use the **Mosquitto** client tools.

Mosquitto is a lightweight, open-source MQTT broker and client implementation widely used for development and testing. The Mosquitto client utilities (mosquitto_pub and mosquitto_sub) allow messages to be published and subscribed to from the command line, making them ideal for verifying MQTT connectivity and message flow.

3.1. Install Mosquitto Client Tools

Download and install the Mosquitto client tools.

On Windows systems, Mosquitto installs by default to:

C:\Program Files\Mosquitto

The Mosquitto command-line tools (mosquitto_pub, mosquitto_sub, etc.) must be accessible to the Command Prompt. There are three ways to do this:

1. Run commands from the installation directory

   cd "C:\Program Files\Mosquitto" -> mosquitto_sub -h test.mosquitto.org -t "senquip/orb1/telemetry" -v

2. Use the full executable path

   "C:\Program Files\Mosquitto\mosquitto_sub.exe" -h test.mosquitto.org -t "senquip/orb1/telemetry" -v

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

3. Add the Mosquitto directory to the Windows PATH environment variable

   Adding the installation folder to the system PATH allows Mosquitto commands to be run from any directory.

3.2. Subscribe from a PC

This procedure uses the public broker:

For this demonstration, use the public broker:

- **Broker:** test.mosquitto.org
- **Port:** 1883
- **TLS:** Disabled
- **Authentication:** None

Subscribe to the telemetry topic:

```
C:\Program Files\Mosquitto\mosquitto_sub.exe" -h test.mosquitto.org -t "senquip/orb1/telemetry" -v
```

Parameters:

- -h Host (broker address)
- -t Topic
- -v Verbose (shows topic with message)

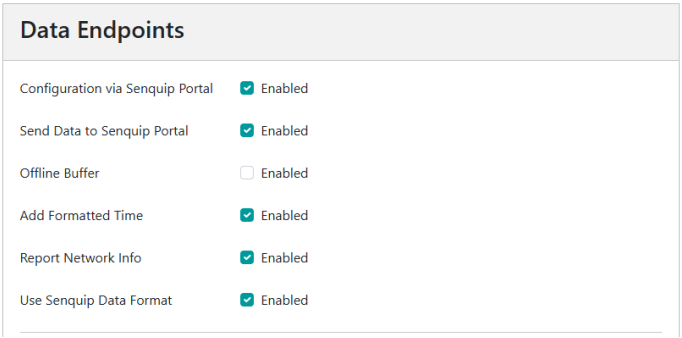
The PC is now subscribed and ready to receive messages.

Figure 5 - PC Subscribed and Waiting for Messages

3.3. Configure a Senquip Device to Publish

Configure the Senquip device to publish standard data to the test broker.

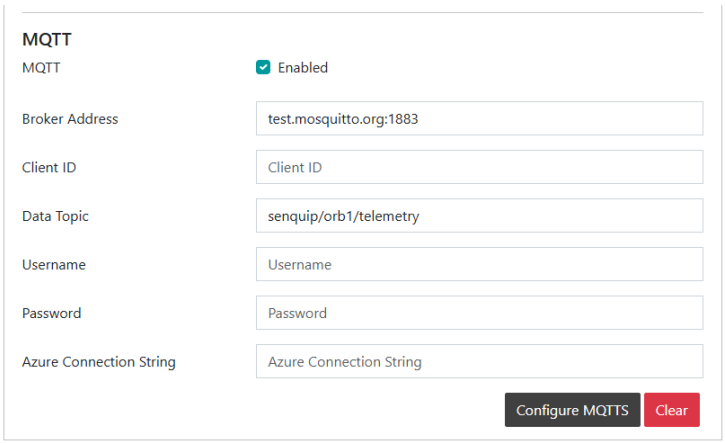
1. Ensure *Use Senquip Data Format* is enabled. This sends the standard Senquip JSON message at each base interval.



Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

2. Enable the MQTT endpoint and configure:

- Broker: test.mosquitto.org
- Port: 1883
- Client ID: leave blank to default to the Device ID
- Authentication: not required



Once configured, the device will publish data at each base interval. Messages should appear in the subscribed PC window.

If no messages appear, refer to Section 2.6 (Testing for Firewalls).

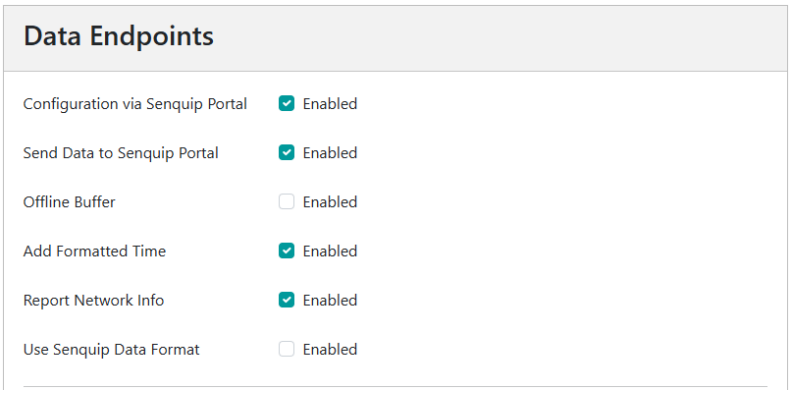
Figure 6 - Published Senquip Data Messages

3.4. Publish a Custom MQTT message from Within a Script

To publish a custom MQTT message instead of the standard data packet:

1. Disable *Use Senquip Data Format*.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.



2. Add a script that publishes the desired data (for example, ambient temperature) to: senquip/orb1/telemetry.

```
load('senquip.js');
load('api_endpoint.js');

SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);
  MQTT.pub('senquip/orb1/telemetry', 'Ambient Temperature: ' + SQ.to_fixed(obj.ambient, 2));
}, null);
```

The custom message should now appear in the subscribed PC window.

```
C:\Program Files (x86)\Mosquitto>mosquitto_sub -h test.mosquitto.org -t "senquip/orb1/telemetry" -v
senquip/orb1/telemetry Ambient Temperature: 36.50
senquip/orb1/telemetry Ambient Temperature: 37.00
senquip/orb1/telemetry Ambient Temperature: 36.50
senquip/orb1/telemetry Ambient Temperature: 36.50
senquip/orb1/telemetry Ambient Temperature: 36.50
```

Figure 7 - Custom Message Arriving

4. Subscribe a Senquip Device to a Topic

This section demonstrates how to subscribe a Senquip device to a topic and how to receive commands can be used to control the Senquip device.

The device will subscribe to: *senquip/orb1/control*

Messages will be JSON formatted:

{state: 'ON'}

{State: 'OFF'}

The device script will:

- Switch the output ON or OFF in response to received messages
- Send state feedback to the Senquip Portal

4.1. Configure Device Subscription

Subscription is implemented within the device script using the previously configured MQTT endpoint settings.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

```
load('senquip.js');
load('api_endpoint.js');

// Subscribe to the topic: 'senquip/orb1/control'
MQTT.sub('senquip/orb1/control', function(conn, topic, msg){
  let obj = JSON.parse(msg);

  SQ.dispatch(20,obj.state); // Send the current state to the Senquip Portal

  if (obj.state === 'ON'){
    SQ.set_output(1, SQ.ON, 0); // turn the output on
    MQTT.pub('senquip/orb1/telemetry', 'State: On'); // Publish the on state
  }
  else if (obj.state === 'OFF'){
    SQ.set_output(1, SQ.OFF, 0); // turn the output off
    MQTT.pub('senquip/orb1/telemetry', 'State: Off'); // Publish the off state
  }
}, null);

SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);
  MQTT.pub('senquip/orb1/telemetry', 'Ambient Temperature: ' + SQ.to_fixed(obj.ambient, 2));
}, null);
```

4.2. Send Commands from the PC

Use the following commands to publish control messages:

ON:

```
"C:\Program Files\Mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "senquip/orb1/control" -m "{\"state\":\"ON\"}"
```

OFF:

```
"C:\Program Files\Mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "senquip/orb1/control" -m "{\"state\":\"OFF\"}"
```

Parameters:

- -h Host
- -t Topic
- -m Message

As messages are published to the control topic, the Senquip device should respond by changing the state of the output, publishing the new state to the telemetry topic, and sending the new state to the Senquip Portal.

```
C:\>"C:\Program Files\Mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "senquip/orb1/control" -m "{\"state\":\"ON\"}"

C:\>"C:\Program Files\Mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "senquip/orb1/control" -m "{\"state\":\"OFF\"}"
```

Figure 8 - On and Off Commands Published to the Control Topic

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

```
C:\Program Files (x86)\Mosquitto>mosquitto_sub -h test.mosquitto.org -t "senquip/orb1/telemetry" -v
senquip/orb1/telemetry Ambient Temperature: 37.00
senquip/orb1/telemetry Ambient Temperature: 36.50
senquip/orb1/telemetry State: On
senquip/orb1/telemetry Ambient Temperature: 36.50
senquip/orb1/telemetry State: Off
senquip/orb1/telemetry Ambient Temperature: 37.00
```

Figure 9 - State Information Arriving on the Telemetry Topic

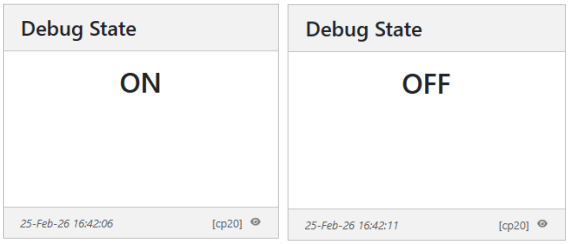


Figure 10 - Status Shown on the Senquip Portal

5. Working With a Secure Connection

Sections 3 and 4 demonstrated MQTT connectivity without encryption. The following sections extend this to production-grade secure configurations using Transport Layer Security (TLS).

TLS provides:

- Encryption of data in transit
- Verification of broker identity
- Optional client authentication
- Protection against interception and impersonation

Two common secure deployment models are used:

1. TLS with server authentication only
2. Mutual TLS (mTLS) with device certificate authentication

5.1. TLS Encryption - Server Authentication Only

In this configuration:

- Communication is encrypted.
- The device verifies the identity of the broker.
- The broker does not authenticate the device using a certificate.
- Optional username/password authentication may still be used.

This model is suitable for development or lower-risk deployments but is not recommended for large fleets or regulated environments without additional controls.

5.1.1. Obtain the Broker CA Certificate

To validate the broker's certificate, the client must trust the Certificate Authority (CA) that issued it.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

In production systems, the broker administrator will provide the required CA certificate. For this demonstration, the CA certificate is obtained from the Mosquitto website.

The CA certificate must be installed on:

- The PC (for mosquitto_pub / mosquitto_sub)
- The Senquip device

Note:

The TLS certificate used by the MQTT listener (port 8883) may differ from the certificate used by the broker's HTTPS website.

For example:

- https://test.mosquitto.org may use a publicly trusted CA.
- mqtts://test.mosquitto.org:8883 may use a Mosquitto-managed CA.

Always verify which CA issued the certificate for the MQTT port.

5.1.2. Configuring the Senquip Device

To enable TLS:

1. Re-enable *Use Senquip Data Format*
2. Set the MQTT port to 8883.
3. Upload the mosquitto.org.crt CA certificate to the device

Once configured:

- All MQTT traffic is encrypted.
- Passive interception is prevented.
- The device verifies it is communicating with the intended broker.

At this stage, communication is encrypted and the broker is verified, but the device itself has not been authenticated using a certificate.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

5.1.3. Subscribe to the Secure Broker

From the command line, subscribe to a topic on the secure Mosquitto broker, noting the location of the CA certificate.

```
"C:\Program Files\Mosquitto\mosquitto_sub.exe" -h test.mosquitto.org -p 8883 --cafile "C:\certs\mosquitto.org.crt" -t "senquip/orb1/telemetry" -v
```

Parameters:

- -h Broker address
- -p Port (8883 for TLS)
- --cafile CA certificate file
- -t Topic
- -v Verbose output

If correctly configured, data will begin appearing in the console.

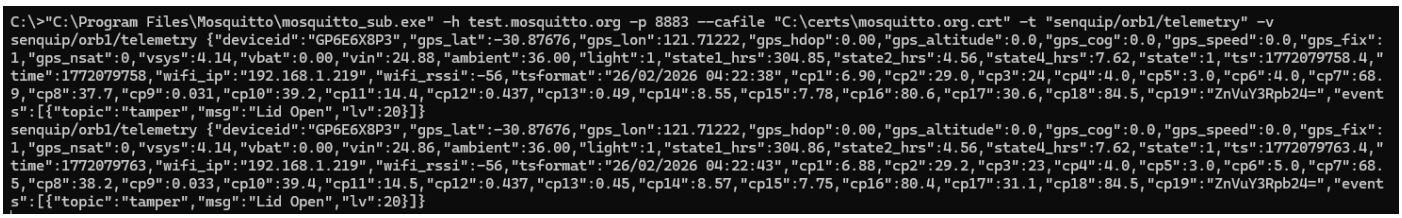


Figure 11 – TLS Encrypted Messages Published by the Senquip Device Arriving on the Telemetry Topic

We now have a device publishing to a secure broker with:

- Encryption of all MQTT traffic
- Protection against passive network interception
- Verification that the device is communicating with the intended broker

However, device identity is not strongly enforced unless additional authentication mechanisms are configured.

6. Mutual TLS (mTLS) – Device Certificate Authentication

Mutual TLS (mTLS) provides the highest level of MQTT security. Senquip uses mTLS for communication with the Senquip Portal.

In an mTLS deployment:

- The device verifies the identity of the broker.
- The broker verifies the identity of the device.
- The TLS session succeeds only if both sides trust each other.

Each device must be provisioned with its own unique X.509 client certificate and private key.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

6.1. Why Mutual TLS Cannot Be Demonstrated Using a Public Test Broker

Unlike basic TLS encryption, mutual TLS requires the broker to validate the device's client certificate against a trusted Certificate Authority (CA).

The list of trusted CAs is configured on the broker and is not negotiated during connection setup. Users of public MQTT test brokers do not have administrative control over this trust configuration.

As a result, you cannot generate your own client certificate and use it with a shared public broker unless the broker operator has explicitly configured the broker to trust the issuing CA or has issued a certificate to you directly.

In practical terms:

- You cannot generate a random client certificate and use it with a public broker.
- The broker must be configured to trust the CA that signed that certificate.
- Only the broker administrator can define which Certificate Authorities are trusted for client authentication.

This requirement highlights a key architectural consideration: when IoT devices connect directly to third-party endpoints, certificate trust and lifecycle management become the responsibility of the endpoint owner.

6.2. What Is Required to Use Mutual TLS with a Third-Party Broker

To connect a Senquip device to a broker using mTLS, the following must be provisioned:

On the broker:

- A trusted Certificate Authority (CA)
- A server certificate signed by that CA
- A configuration that enforces client certificate validation

On each Senquip device:

- The broker's CA certificate (to verify the broker)
- A unique client certificate (client.crt)
- The corresponding private key (client.key)

Each client certificate must:

- Be securely generated
- Be securely transferred to the device
- Be uniquely assigned to that device
- Be tracked for expiry
- Be revoked if compromised

This process must be repeated for every device in the fleet.

6.3. Operational Considerations

Managing individual device certificates introduces operational complexity:

- Secure key generation procedures

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

- Secure storage of private keys
- Certificate distribution processes
- Renewal and expiry management
- Revocation capability
- Audit logging and traceability

In small deployments this may be manageable.

In large fleets, certificate lifecycle management becomes a significant administrative and security responsibility.

This is often underestimated during initial system design.

6.4. Architectural Implications

When connecting directly to a third-party MQTT endpoint, the customer assumes responsibility for:

- Broker security configuration
- Certificate Authority management
- Device certificate provisioning
- Ongoing certificate lifecycle management

These tasks require secure procedures and technical expertise.

Managing individual certificates for each device is complex, time-consuming, and requires robust security controls.

For data sent to the Senquip Portal, Senquip manages these processes in accordance with our Information Security Management System (ISMS).

The hidden operational cost of maintaining a secure MQTT infrastructure often exceeds the perceived benefit of direct device-to-customer server integration.

For this reason, it is frequently more cost-effective to:

1. Allow devices to securely publish data to the Senquip Portal.
2. Retrieve required data via the Senquip API.
3. Integrate downstream systems using controlled, server-side access.

This approach centralises certificate management, reduces risk, and simplifies fleet administration.

7. Summary

Using Mosquitto with a public broker provides a simple and reliable method to verify MQTT connectivity, publish operation, and subscription handling on Senquip devices.

Production grade MQTT deployments require encryption, broker verification, and, in many cases, mutual TLS with per-device certificates. Implementing and maintaining mutual TLS introduces certificate provisioning, secure key storage, renewal, expiry management, and revocation processes for every device in the fleet.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

When connecting devices directly to a third-party MQTT endpoint, these responsibilities rest with the endpoint owner. At scale, this represents a non-trivial operational and security burden that must be properly designed and maintained.

For this reason, many deployments choose to publish data securely to the Senquip Portal, where certificate management, encryption, and broker security are centrally managed under Senquip's Information Security Management System (ISMS). Where integration with external systems is required, data can be retrieved securely via the Senquip API.

Centralising certificate management at the platform level reduces risk, simplifies fleet administration, and avoids the complexity of managing device certificates across distributed endpoints.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.