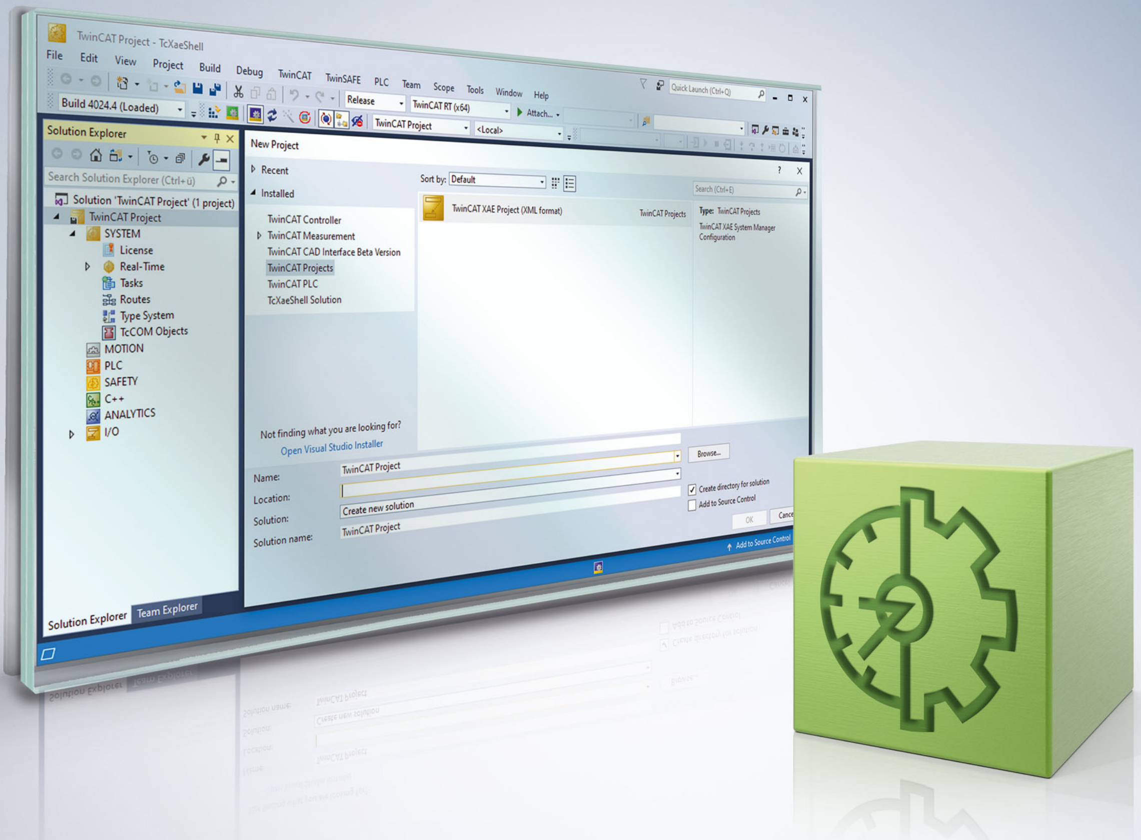
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | ADS-over-MQTT



2023-11-29 | Version: 1.3.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

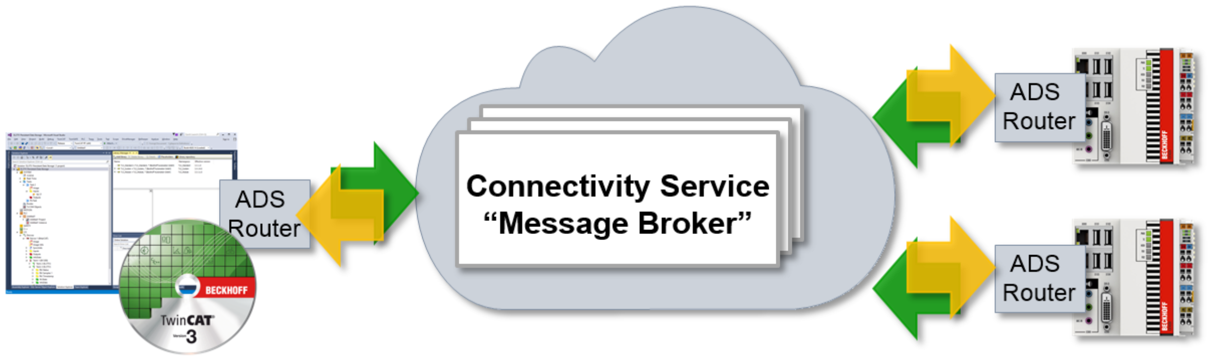
Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 General description

From the point of view of the ADS protocol, "ADS-over-MQTT" is a new transport channel. This means that precisely the same ADS commands are transmitted over MQTT as over other communication protocols.

To do this the TwinCAT router establishes a connection to the broker in order to send and also receive ADS protocol commands.
The end point of the broker is thus configured on the local device. The result of this is that the 1:1 relationship of an ADS route is only created in the interaction with the matching broker.



This document provides an overview of the usage possibilities as well as a technical description of how a "virtual ADS network" can be configured over an MQTT message broker.

Benefits of an MQTT-based ADS network

- **Subnets, NAT-based networks and firewalls:**
  Incoming TCP/IP connections are used in both directions in a classic ADS setup. This makes it necessary for the devices to be located in the same network in the normal case. In distributed systems with different subnets this leads to complex configurations in order to make the ADS routes usable.
  In the case of MQTT-based ADS networks, only an outgoing TCP/IP connection is used by the devices. This allows the broker in the higher-level network to broker between all devices.
  Due to the outgoing connections, a typical firewall can be used and no incoming ports need to be registered.
- **Access control:**
  After creating the appropriate routes, bidirectional communication can be executed in a classic ADS setup.
  An access by device A, which accesses B, also allows device B to access A.
  The MQTT-based ADS network can be configured so that device A can access B, but not the other way around.
- **Security / encryption:**
  The communication from TwinCAT to the broker can be encrypted by TLS (with certificates or PreSharedKey (PSK)).

The increased administrative effort should be regarded as disadvantageous. However, this would be reduced to a reasonably low level per device in a larger network.

NOTICE

ADS access means full access

As described in Security Advisory 2017-01, ADS offers full access to a device.
Secure ADS offers authorization as well as encryption for the communication; therefore, it represents a transport encryption. Hence, if an ADS route exists, then full access exists.
Dedicated, role-related access to individual files is offered by solutions such as OPC-UA.

3 Requirements

TwinCAT 3.1 build 4022.0 required

ADS-over-MQTT is an extension of build 4022 and therefore only available from this release.

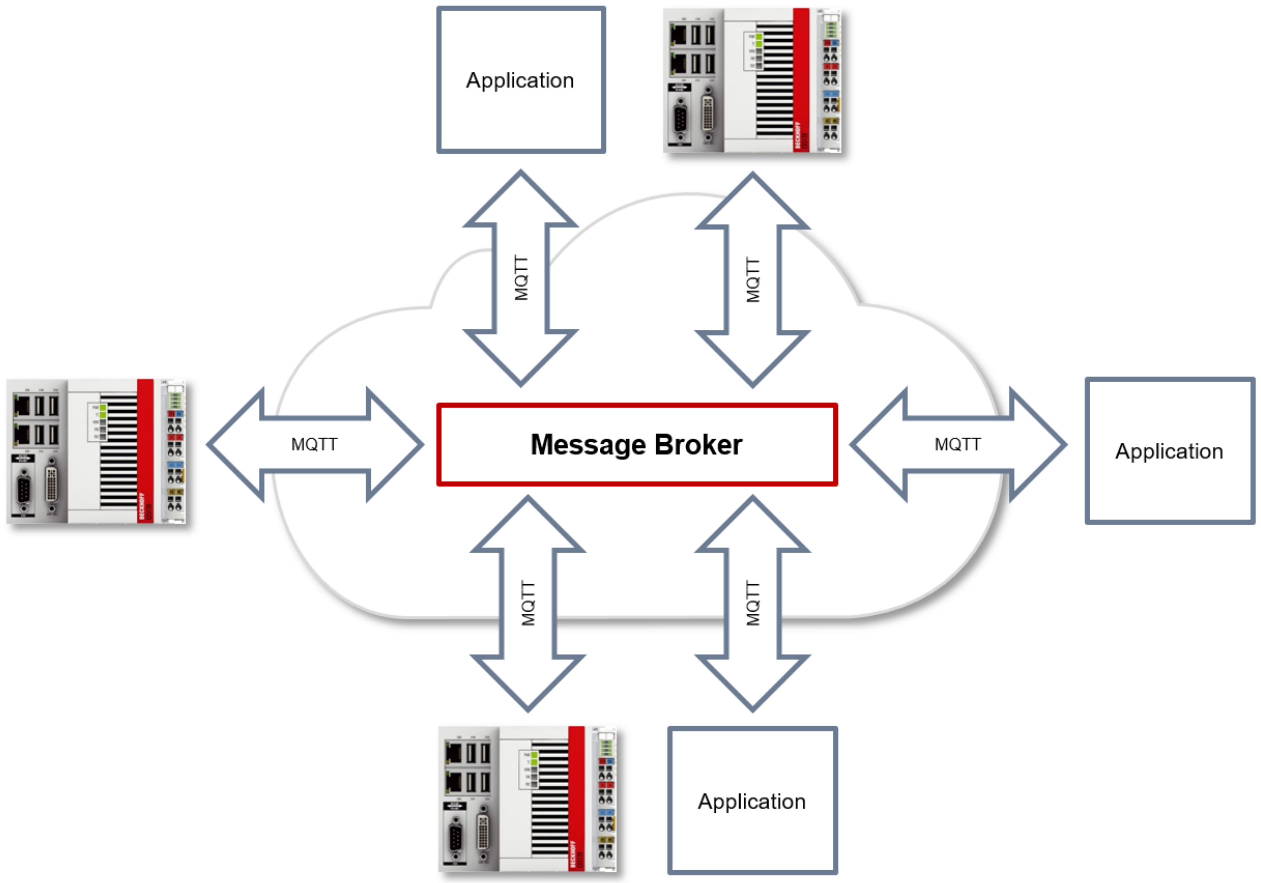
- ADS-over-MQTT is a component of TC1000 and can be used without license costs.
- The devices used need outgoing network communication to the broker.
- An MQTT broker must be provided via which the communication can take place.
- The extension provided is available for the Eclipse Mosquitto broker.
- Appropriate certificates may need to be generated and signed for TLS encryption.

4 Technical introduction

This section provides an overview of the technologies used as well as the basic architecture of a "virtual ADS network".
ADS-over-MQTT introduces an additional communication channel for this, resulting in ADS routes over MQTT. This can use the programs started as ADS devices on the devices without them being modified.

4.1 MQTT basics

MQTT (Message Queueing Telemetry Transport) is a publisher/subscriber-based communication protocol which enables message-based transfer between applications. The message broker is a central component of this transfer type. It distributes messages between the individual applications or the sender and receiver of a message. The message broker decouples the sender and receiver, so that it is not necessary for the sender and receiver to know and exchange each other's address information. During sending and receiving, all communication devices contact the message broker, which handles the distribution of the messages.



ClientID

When establishing a connection with the message broker, the client transmits a ClientID, which is used to uniquely identify the client on the message broker. The MQTT communication driver from TwinCAT 3 automatically generates its own ClientID, which is based on the following naming scheme:

PlcProjectName-TcMqttClient%n

%n is an incremental counter for the number of the respective MQTT client instance. Each instance of the FB_IotMqttClient function block increments this counter. In most cases, using this ClientID format is sufficient. In special cases, e.g. depending on the message broker or also due to the own MQTT application, an application-specific ClientID must be assigned. This can be done via a corresponding input at the FB_IotMqttClient and FB_IotMqtt5Client function blocks.

If a unique ClientID is to be generated automatically at the start of the PLC project, the use of a GUID is recommended, which can be generated via the FB_CreateGuid function block from the Tc2_System library. The following sample code illustrates the use of this function block.

```
PROGRAM MAIN
VAR
  fbGuid : FB_CreateGUID;
  objGuid : GUID;
  sGuid : STRING;
  nState : UINT;
  bStart : BOOL; // set to TRUE to start this sample
END_VAR

CASE nState OF
  0 :
    IF bStart THEN
      bStart := FALSE;
      nState := nState + 1;
    END_IF

  1 : // create GUID using FB_CreateGuid from Tc2_System library
    fbGuid(bExecute := TRUE, pGuidBuffer := ADR(objGuid), nGuidBufferSize := SIZEOF(objGuid));
    IF NOT fbGuid.bBusy THEN
      fbGuid(bExecute := FALSE);
      IF NOT fbGuid.bError THEN
        nState := nState + 1;
      ELSE
        nState := 255; // go to error state
      END_IF
    END_IF

  2: // GUID has been created, now convert to STRING
    sGuid := GUID_TO_STRING(objGuid);
    nState := nState + 1;

  3: // done

255: // error state

END_CASE
```

After execution of this State Machine, the variable sGuid contains the generated GUID as STRING. This can then be used at the FB_IotMqttClient and FB_IotMqtt5Client function blocks as ClientID.

Payload

The content of an MQTT message is referred to as payload. Data of any type can be transferred, e.g. text, individual numerical values or a whole information structure.

Message payload formatting

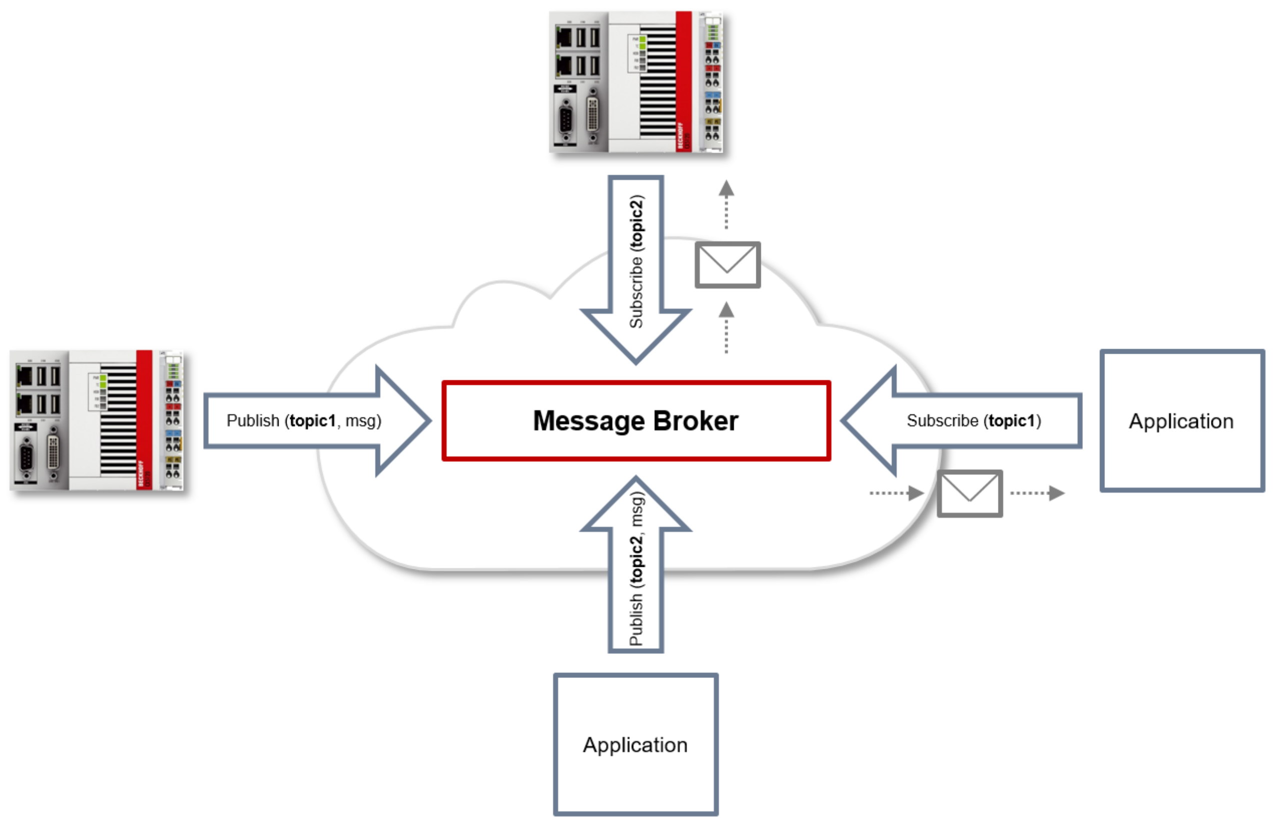
Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

Topics

If a message broker is used that is based on the MQTT protocol, sending (publish mode) and subscribing (subscribe mode) of messages is organized with the aid of so-called topics. The message broker filters incoming messages based on these topics for each connected client. A topic may consist of several levels; the individual levels are separated by “/”.

Example: Campus / Building1 / Floor2 / Room3 / Temperature

When a publisher sends a message, it always specifies for which topic it is intended. A subscriber indicates which topic it is interested in. The message broker forwards the message accordingly.



Communication example 1 from the diagram above:

- An application subscribes to “topic1”.
- A controller publishes a message to “topic1”.
- The message broker forwards the message to the application accordingly.

Communication example 2 from the diagram above:

- A controller subscribes to “topic2”.
- An application publishes a message to “topic2”.
- The message broker forwards the message to the controller accordingly.

Wildcards

It is possible to use wildcards in conjunction with topics. A wildcard is used to represent part of the topic. In this case a subscriber may receive messages from several topics. A distinction is made between two types of wildcards:

- Single-level wildcards
- Multi-level wildcards

Example for single-level wildcard:

The + symbol describes a single-level wildcard. If it is used by the subscriber as described below, for example, corresponding messages to the topics are either received by the subscriber or not.

- The receiver subscribes to Campus/Building1/Floor2/**+**/Temperature
- The publisher sends to Campus/Building1/Floor2/Room1/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room2/Temperature - OK
- The publisher sends to Campus/Building42/Floor1/Room1/Temperature - NOK
- The publisher sends to Campus/Building1/Floor2/Room1/Fridge/Temperature - NOK

Example for multi-level wildcard:

The # symbol describes a multi-level wildcard. If it is used by the subscriber as described below, for example, corresponding messages to the topics are either received by the subscriber or not. The # symbol must always be the last symbol in a topic string.

- The receiver subscribes to Campus/Building1/Floor2/**#**
- The publisher sends to Campus/Building1/Floor2/Room1/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room2/Temperature - OK
- The publisher sends to Campus/Building42/Floor1/Room1/Temperature - NOK
- The publisher sends to Campus/Building1/Floor2/Room1/Fridge/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room1/Humidity - OK

QoS (Quality of Service)

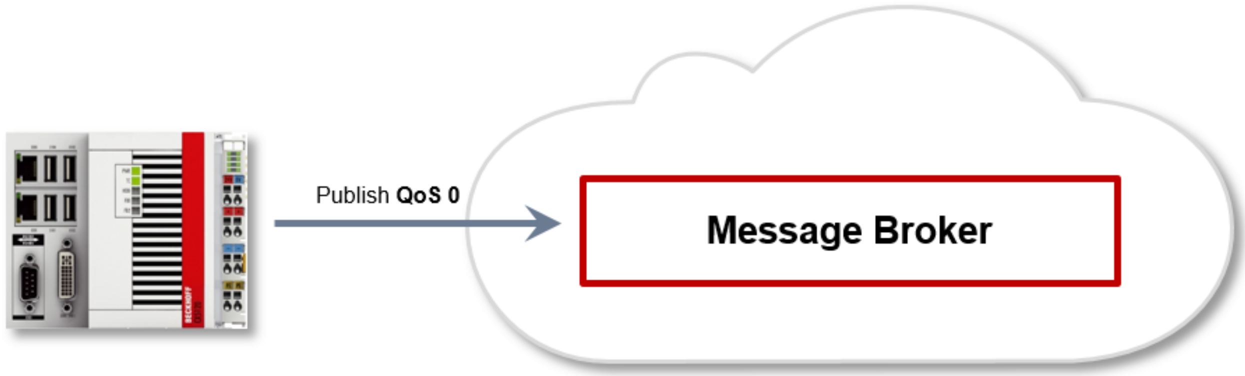
QoS is an arrangement between the sender and receiver of a message with regard to guaranteeing of the message transfer. MQTT features three different levels:

- 0 – not more than once
- 1 – at least once
- 2 – exactly once

Both types of communication (publish/subscribe) with the message broker must be taken into account and considered separately. The QoS level that a client uses for publishing a message is set by the respective client. When the broker forwards the message to client that has subscribed to the topic, the subscriber uses the QoS level that was specified when the subscription was established. This means that a QoS level that may have been specified as 2 by the publisher can be “overwritten” with 0 by the subscriber.

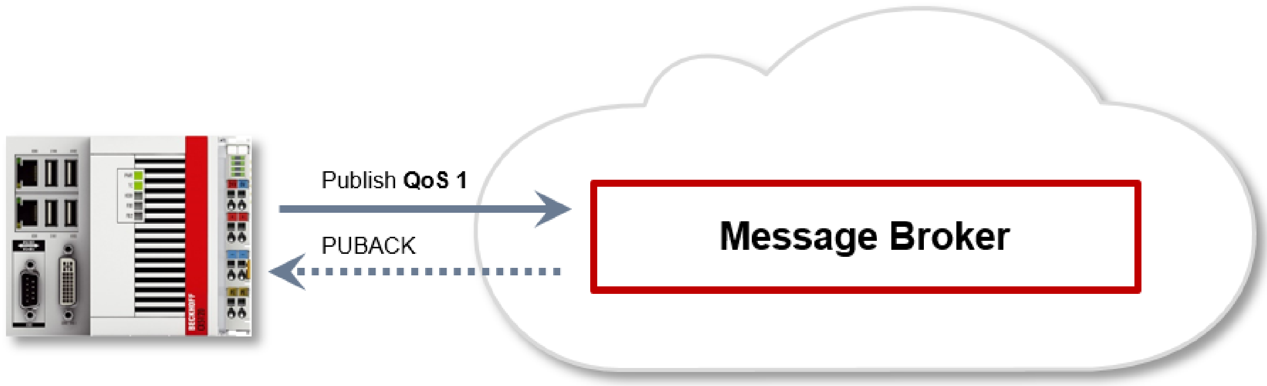
QoS-Level 0

At this QoS level the receiver does not acknowledge receipt. The message is not sent a second time.



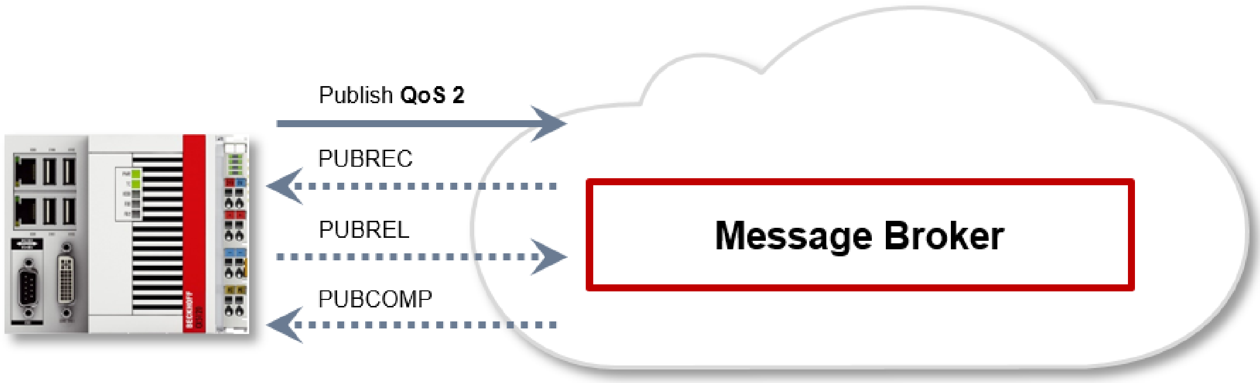
QoS-Level 1

At this QoS level the system guarantees that the message arrives at the receiver at least once, although the message may arrive more than once. The sender stores the message internally until it has received an acknowledgement from the receiver in the form of a PUBACK message. If the PUBACK message fails to arrive within a certain time, the message is resent.



QoS-Level 2

At this QoS level the system guarantees that the message arrives at the receiver no more than once. On the MQTT side this is realized through a handshake mechanism. QoS level 2 is the safest level (from a message transfer perspective), but also the slowest. When a receiver receives a message with QoS level 2, it acknowledges the message with a PUBREC. The sender of the message remembers it internally until it has received a PUBCOMP. This additional handshake (compared with QoS 1) is important for avoiding duplicate transfer of the message. Once the sender of the message receives a PUBREC, it can discard the initial publish information, since it knows that the message was received once by the receiver. In other words, it remembers the PUBREC internally and sends a PUBREL. Once the receiver has received a PUBREL, it can discard the previously remembered states and respond with a PUBCOMP, and vice versa. Whenever a package is lost, the respective communication device is responsible for resending the last message after a certain time.



The LastWill is a message sent by the broker to all clients subscribed to the matching topic in the event of an abnormal connection failure. If the MQTT client in the PLC loses the connection to the broker and a LastWill was stored when the connection was established, this LastWill is communicated by the broker without the client having to do it.

In the event of a planned disconnect, the LastWill is not necessarily transmitted according to the specification. From the PLC programmer's point of view, he can decide whether he wants to publish the LastWill before calling the disconnect. To this end, the LastWill message is published again on the LastWill topic. This is necessary because the broker would not publish the LastWill message due to the regular disconnection.

In the event of a TwinCAT context change and a resulting restart of the MQTT communication, the IoT driver sends the previously specified LastWill to the broker, because at this point, doing this from the PLC is not an option. If no LastWill was defined when the connection was established, no message will be transmitted before the disconnect.

Safety

When a connection to the message broker is established, security mechanisms such as TLS can be used to encrypt the communication connection or to execute authentication between client and message broker.

Sources

For further and more detailed information about MQTT, we recommend the following sites:

HiveMq Blog: http://www.hivemq.com/blog/mqtt-essentials/ (the main basis for this article)

4.2 Architecture

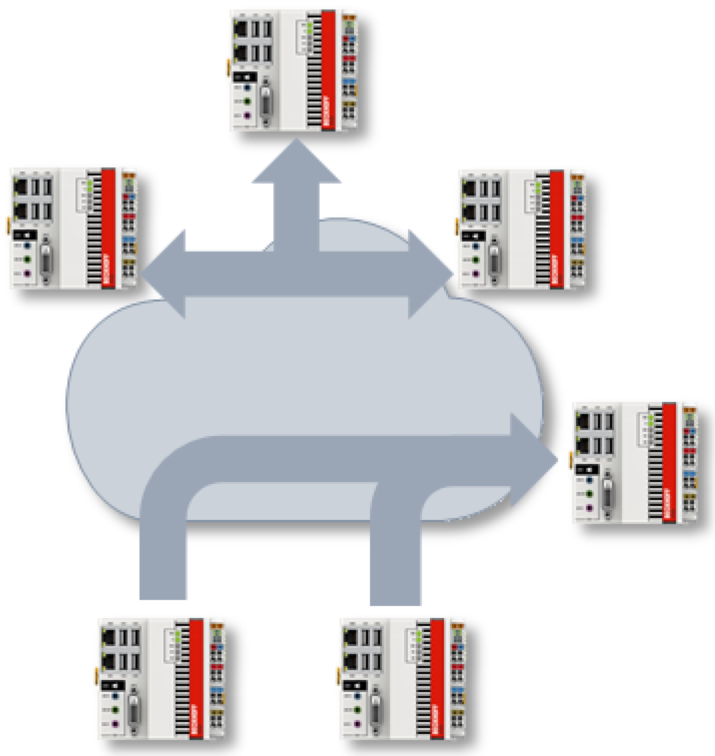
The ADS router in each device brokers the ADS commands between the local and also remote "ADS devices".
This router can be configured so that ADS communication can also take place over a broker.
The broker brokers the incoming ADS commands on the basis of the stored configuration.

Virtual AMS network

Different "virtual AMS networks" with different devices can be defined in the broker. To do this, each TwinCAT router opens an MQTT connection to the broker that is set in its configuration.

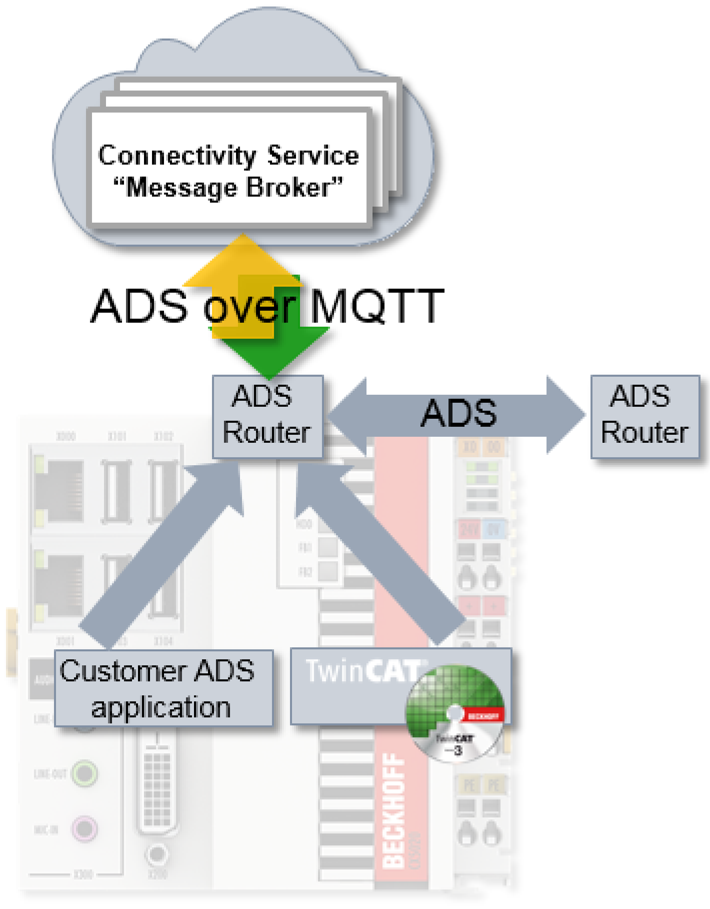
The broker configuration specifies which devices are permitted to access which other devices.

Overall, virtual AMS networks can be mapped via a broker.



Local realization

The realization of the ADS-over-MQTT connection takes place via the TwinCAT router as an additional transport channel. As a result, the extension is transparent with regard to the ADS client as well as the ADS servers on the respective devices.



Technical realization

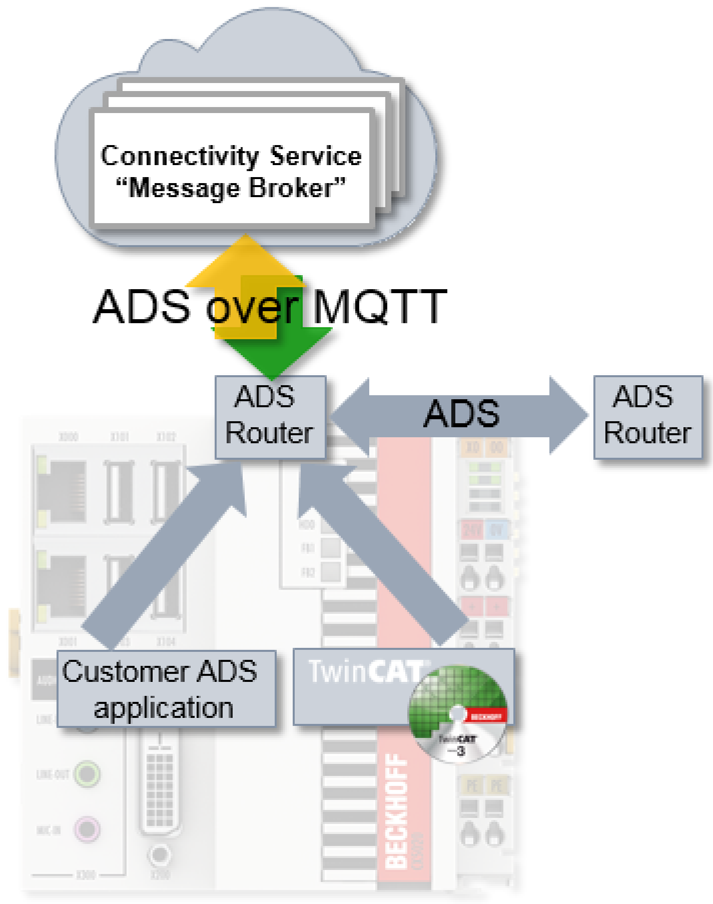
At MQTT protocol level each ADS router is mapped as a "user", although this need not represent an exclusive relationship.
Two different topics categories are used by each communication device:

- Discovery: **<NetworkName>/<AmsNetId>/info**
  A connecting router sends a **RETAIN** message to this topic whilst at the same time subscribing to **<NetworkName>/+/info (QoS2)** so that it is informed about other connected routers.
- Communication: **<NetworkName>/<AmsNetId>/ams/#**
  A router subscribes to **<NetworkName>/<AmsNetId>/ams/# (QoS2)**.
  The ADS commands are sent to this router at **<NetworkName>/<AmsNetId>/ams** and the responses via **<NetworkName>/<AmsNetId>/ams/res.**

The result of this is that the broker has to implement **RETAIN** topics as well as **QoS**, as described in the introduction. One example of this is the Eclipse Mosquitto broker.

4.3 Transparent retrofitting

The realization of ADS-over-MQTT inside the TwinCAT router makes the retrofitting of applications possible. None of the ADS applications (client and server) – this also includes applications written by the customer – need to be recompiled.



The ADS applications use ADS routes to identify the communication partner. This ADS route is independent of the transport channel and is described in the TwinCAT router.

If the route used is switched to an ADS-over-MQTT connection, the ADS traffic is transported via the broker (and thus secured if necessary).

5 Configuration

The configuration is done using XML files both on the TwinCAT system side and for the MQTT broker.

5.1 TwinCAT

The TwinCAT router is configured by an XML in order to establish a connection with one or more routers.

To do this the XML files described here can be saved with any desired name in the folder *C:\TwinCAT\3.x\Target\Routes* (**Windows CE: \Hard Disk\TwinCAT\3.x\Target\Routes\**) (x = TwinCAT version number). Saved changes are accepted when the TwinCAT router is initialized, which takes place, for example, during the transition RUN->CONFIG or CONFIG->CONFIG.

The XML file has the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/TcConfig">
 <RemoteConnections>
  <Mqtt>
   <Address Port="1883">BROKER-ADDRESS</Address>
   <Topic>VirtualAmsNetwork1</Topic>
   <User>CX-123456</User>
  </Mqtt>
 </RemoteConnections>
</TcConfig>
```

A connection is established for this and the TwinCAT router logs onto the broker, which is reachable via BROKER-ADDRESS, with the given name (in this case CX-123456) and the port 1883. The BROKER-ADDRESS is thereby the IP or name of the computer on which the broker is running.

The TwinCAT router is at the same time a device on the network "VirtualAmsNetwork1" in the broker, which is reflected in the topics used as described in Architecture [▶ 16].
The <User> element thereby specifies the user at MQTT level and can be used in the broker, e.g. in the Broker [▶ 18], to configure accesses.
Optionally, the <Mqtt> element can carry an attribute, ClientId, in order to specify the MQTT ClientId. This is otherwise formed from the <User> and an arbitrary string.

This configuration establishes an unencrypted connection; encryption options are documented under Security [▶ 20].

5.2 Broker

The MQTT broker is used to broker the ADS commands between the routers. The topic structure used is described in Architecture [▶ 16].

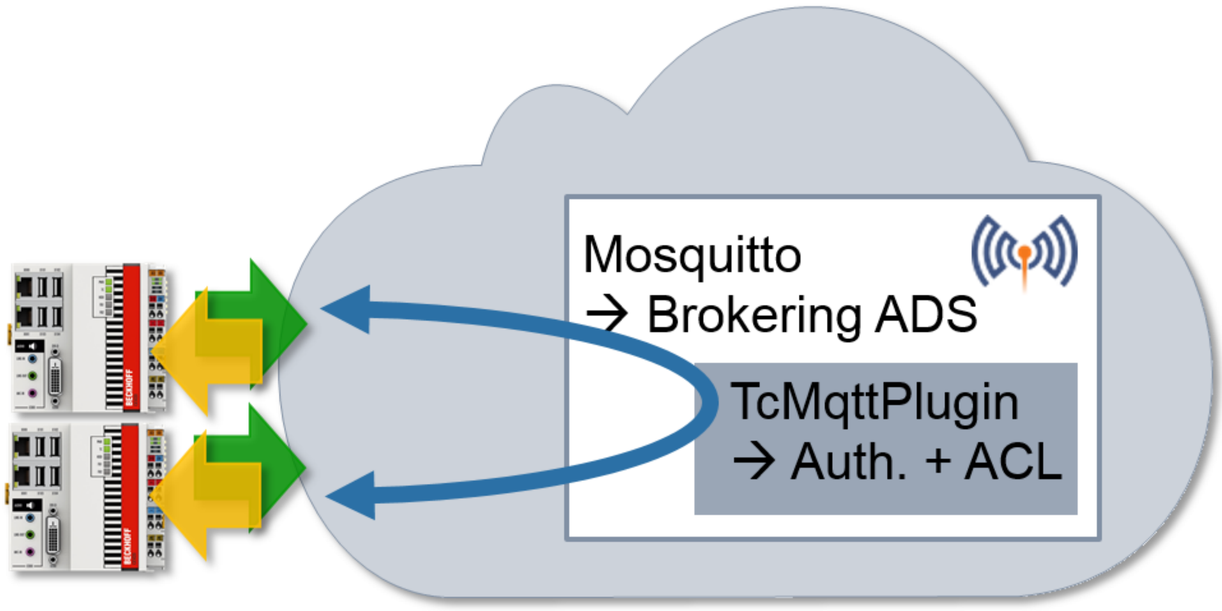
General

Any MQTT broker can be used for ADS-over-MQTT with suitable support of, for example, RETAIN and QoS.

Appropriate measures must be taken if this broker needs to be protected in terms of security because the ADS messages need to be protected. The security configuration on the TwinCAT side and, for example, for the Eclipse Mosquitto Broker is described in Security [▶ 20].

Tc-Plugin TcMqttPlugin.dll for the Eclipse Mosquitto Broker

In order to define a virtual network of ADS devices in the MQTT broker, there is an extension for the Eclipse Mosquitto Broker. Using this extension, access rights can be set by PreSharedKey on the broker and accesses between the TwinCAT routers can be set by means of an ACL (AccessControlList).



The plugin is supplied with the TwinCAT installation and is located in the folder *C:\TwinCAT\AdsApi\TcMqttPlugin* or *C:\TwinCAT\AdsApi\x64\TcMqttPlugin* if a 64-bit Mosquitto Broker is used.
The plugin is integrated in the Mosquitto configuration as follows:

```
auth_plugin <Path>TcMqttPlugin.dll
auth_opt_xml_file <Path>ACL.xml
```

The Mosquitto configuration file is specified when starting the Mosquitto broker by means of the parameter "-c", which loads the plugin including the configuration.

The file ACL.xml is thereby described in the following sections and provides the access configuration by PreSharedKey on the broker itself as well as the configuration of the communication between the connected TwinCAT routers.

Configuration of "Virtual Ams Network"

The plugin TcMqttPlugin offers the option of configuring virtual Ams networks. To do this, specify which device can access which other device for each target device.
Unlike classic ADS routes, these connections are directional: A target therefore has no right at the same time to access the source.

```
<TcMqttAclConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\TwinCAT\3.1\Config\Modules\TcMqttAclConfig.xsd"
AnonymousLogin="true">
<!-- PSK Elements, if used -->
<Ams>
<Topic>VirtualAmsNetwork1</Topic>
<User>
<Name>EngineeringStation</Name>
</User>
<User>
<Name>CX-123456</Name>
<Access>EngineeringStation</Access>
</User>
<User>
<Name>CX-567890</Name>
<Access>EngineeringStation</Access>
</User>
</Ams>
</TcMqttAclConfig>
```

The name of the Ams network is defined within an <Ams> node. It is used in the MQTT topics employed for the identification of the networks.
Individual <User> elements describe the devices. These elements have a <Name> element that describes the MQTT identity with which the connection was established – in the normal case the name of the device. In

addition, access-entitled devices are defined via the <Access> element.
In the example, "EngineeringStation" can thus access two CX devices, but the CX devices can access neither the "EngineeringStation" nor each other.

The file is cyclically reloaded so that a broker restart is unnecessary.

As no encryption is foreseen in this explanation, `AnonymousLogin=“true“` is used.

Restrictions with regard to the AmsNetId to be registered

With this configuration each validly connected device can assume an arbitrary AmsNetID and thus an identity from the point of view of ADS. This can be restricted:

```
<User>
  <Name>CX-567890</Name>
  <Access>EngineeringStation</Access>
  <NetId>192.168.56.1.1.1</NetId>
</User>
```

As soon as at least one NetId is specified, only one NetId can be registered from this list.

Mosquitto settings

In connection with the configuration by means of TcMqttPlugin, it is important to observe some of the settings on the Mosquitto Broker side. These include:

- `psk_hint` Designates the psk_hint for the establishment of the connection. Not currently checked on the TwinCAT side.
- `port <1883|8883>` The port designates the network port provided by the broker. Typically, 1883 is unencrypted and 8883 encrypted.
- `require_certificate <true|false>` Indicates the necessity of certificates.
- `use_identity_as_username true` Indicates whether the identity is used by certificates as a user name at MQTT level. This is used in order to use the TcMqttPlugin, therefore it must be set to `true`.

Minimum configuration examples are described in the corresponding sections according to the TLS connection used.

5.3 Security

There are options for securing the communication. A TLS connection on the basis of X.509 certificates or a PreSharedKey (PSK) can be used for this.

It is recommended that communication be secured with TLS especially when communicating over non-trustworthy networks (e.g. the Internet). The broker itself must be operated in a trustworthy environment, since all messages are unsecured there.

Compromising of the virtual ADS network

Even when communication between the devices and the broker takes place in encrypted form via TLS, the devices are not secured among one another. The ADS commands are present on the broker in unencrypted form.
If a device is compromised, the attacker can execute all ADS commands via the rights gained. These commands also include file reading operations or operations for starting processes.

5.3.1 TLS / PreSharedKey (PSK)

PreSharedKeys (PSK) are passwords that are applied on both sides of a connection through a configuration process. A TLS 1.2 connection is used for communication.

TwinCAT configuration with PSK

For a TwinCAT router a PSK can be applied to the route in the configuration file, wherein the key is entered as a hex string.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/TcConfig">
<RemoteConnections>
  <Mqtt>
    <Address Port="8883">BROKER-ADDRESS</Address>
    <Topic>VirtualAmsNetwork1</Topic>
    <Psk>
      <Identity>EngineeringStation</Identity>
      <Key>4D65696E5061737377C3B67274[…]</Key>
    </Psk>
  </Mqtt>
</RemoteConnections>
</TcConfig>
```

Secure PSK

A meaningful PreSharedKey is formed from a hex string of 64 characters.

Alternatively, the key can also be determined by TwinCAT to allow simpler input. To do this a password is entered as a normal string in the <Pwd> element. TwinCAT calculates the PSK to be used from this and the identity by means by Sha256('Identity'+'Pwd'). If the attribute "IdentityCaseSensitive" is set to "false" (or not), the identity is used as an upper-case string for the key calculation.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/TcConfig">
<RemoteConnections>
  <Mqtt>
    <Address Port="8883">BROKER</Address>
    <Topic>VirtualAmsNetwork1</Topic>
    <Psk>
      <Identity>EngineeringStation</Identity>
      <Pwd IdentityCaseSensitive="false">!ABCDEFGHijklmn123545</Pwd>
    </Psk>
  </Mqtt>
</RemoteConnections>
</TcConfig>
```

Minimal Mosquitto configuration

The following entries can be used for PSKs as the simplest Mosquitto configuration:

```
port 8883
psk_hint AHint
use_identity_as_username true
auth_plugin C:\TwinCAT\AdsApi\TcMqttPlugin\TcMqttPlugin.dll
auth_opt_xml_file ACL.xml
```

Broker configuration with PSK

The TcMqttPlugin offers the option to use a PSK in the broker in order to be able to access a broker. The configuration is saved in the configuration file of the plugin, wherein the PSK is specified as a hex string.

The IdentityCaseSensitive offers the option of regarding the identities irrespective of whether they are written in lower or upper case.

```
<TcMqttAclConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\TwinCAT\3.1\Config\Modules\TcMqttAclConfig.xsd"
IdentityCaseSensitive="false">
  <Psk>
    <Identity>EngineeringStation</Identity>
    <Key>4D65696E5061737377C3B67274</Key>
  </Psk>
<!-- <Ams> Elements following -->
</TcMqttAclConfig>
```

Alternatively, the key can also be determined by the TcMqttPlugin to allow simpler input. To do this a password is entered as a normal string in the <Pwd> element. TwinCAT calculates the PSK to be used from this and the identity by means by Sha256('Identity'+'Pwd'). If the attribute at the level of `<TcMqttAclConfig>` "IdentityCaseSensitive" is set to "false" (or not), the identity is used as an upper-case string for the key calculation.

```
<TcMqttAclConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\TwinCAT\3.1\Config\Modules\TcMqttAclConfig.xsd"
IdentityCaseSensitive="false">
  <Psk>
    <Identity>EngineeringStation</Identity>
    <Pwd>!ABCDEFGHijklmn123545</Pwd>
  </Psk>
<!-- <Ams> Elements following -->
</TcMqttAclConfig>
```

5.3.2 TLS / certificates

Certificates conforming to X.509 standard can be used to secure the corresponding MQTT connection to the broker.

TwinCAT configuration with certificates

For a TwinCAT router the paths to the X.509 certificates can be configured in the MQTT routes:

```
<?xml version="1.0" encoding="UTF-8"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/TcConfig">
<RemoteConnections>
  <Mqtt>
    <Address Port="8883">BROKER-ADDRESS</Address>
    <Topic>VirtualAmsNetwork1</Topic>
    <Tls>
      <Ca>C:\TwinCAT\3.1\Target\Certificates\CA.crt</Ca>
      <Cert>C:\TwinCAT\3.1\Target\Certificates\Device.crt</Cert>
      <Key>C:\TwinCAT\3.1\Target\Certificates\Device.key</Key>
    </Tls>
  </Mqtt>
</RemoteConnections>
</TcConfig>
```

In this case the corresponding paths to the files are entered in the element <Tls>. <Ca> is thereby the X.509 certificate of the Certificate Authority, i.e. the issuing body by whom certificates should be accepted.

The elements <Cert> and <Key> contain paths to the public and private key of the certificate to be used.

- The host name of the broker ("BROKER ADDRESS") must match the Common Name of the certificate used. This is checked by the clients.
- The Common Name of the client certificate is used as Identity in MQTT (and in TcMqtt.dll).

Minimal Mosquitto configuration

The following entries can be used as the simplest Mosquitto configuration for the use of certificates:

```
port 8883
cafile cert/CA.crt
certfile cert/Broker.crt
keyfile cert/Broker.key
require_certificate true
use_identity_as_username true
auth_plugin C:\TwinCAT\AdsApi\TcMqttPlugin\TcMqttPlugin.dll
auth_opt_xml_file ACLCerts.xml
```

Broker configuration with certificates

The identity used in the <Ams> elements to describe the AmsNetwork is defined via the CN of the certificate.

The Certificate Authority defines which certificates are granted access.

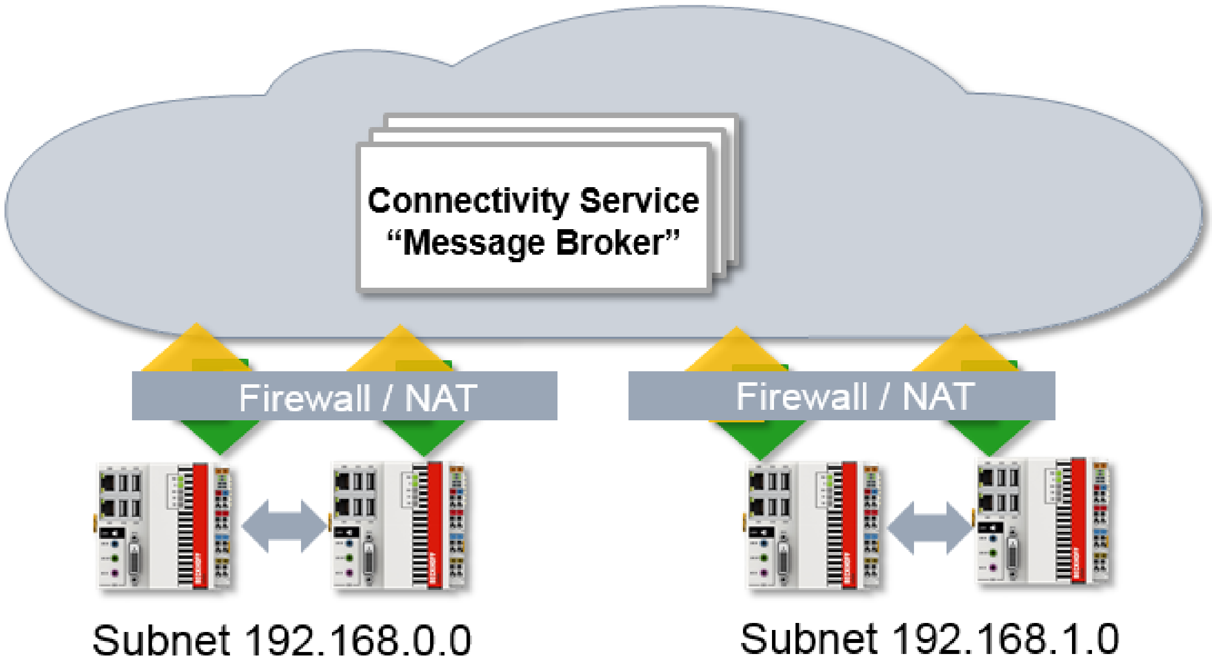
An additional configuration on the broker side is thus unnecessary.

6 Application scenarios

At this point several application scenarios will be described in order to demonstrate the added value of ADS-over-MQTT.

6.1 NAT-based networks

The outgoing MQTT connections from TwinCAT to the broker enable simple communication between subnets: All connected devices must be able to establish an outgoing connection to the broker – this one connection is used for the entire ADS communication. The broker is the only component with incoming connections.
This is particularly advantageous in production processes in large and possibly distributed systems. Subnets with NAT, firewalls, etc. are frequently used here. Nevertheless, an ADS communication beyond the network limits should be enabled from time to time.
Such a communication option is valuable in many cases. However, subnets can also be set up for security reasons so that communication is not desirable (keyword: "zoning").

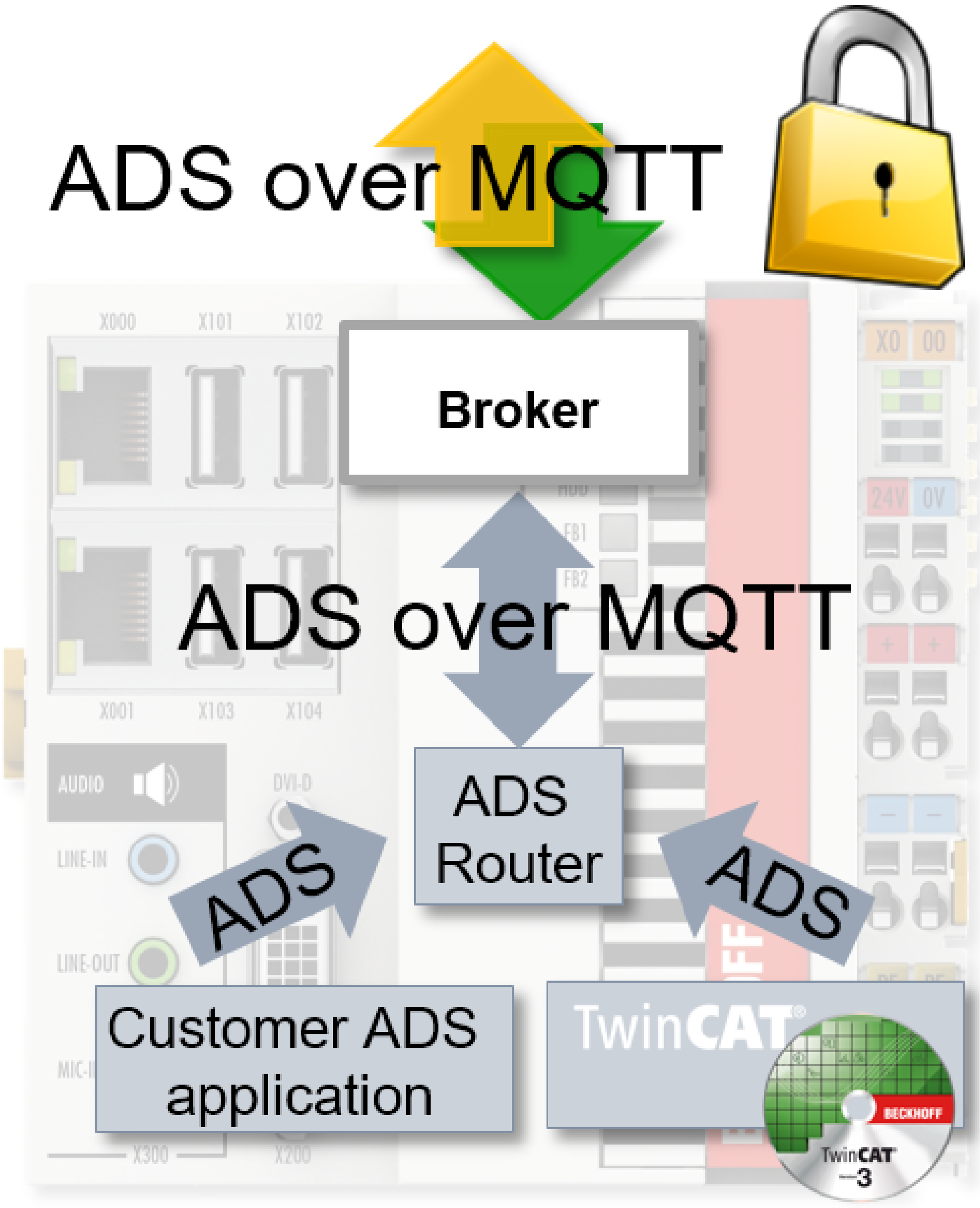


6.2 ADS encryption

MQTT can also be used to enable encrypted ADS communication through the capability of TLS, which is used for the encryption of MQTT at transport level.
The broker can be installed locally on the PC-based controllers for this. In doing so the broker is configured so that it merely offers the local controller as an access point in a virtual Ams Network.

If encryption is then activated and used, this creates a TLS-based secured connection via ADS to a TwinCAT system.

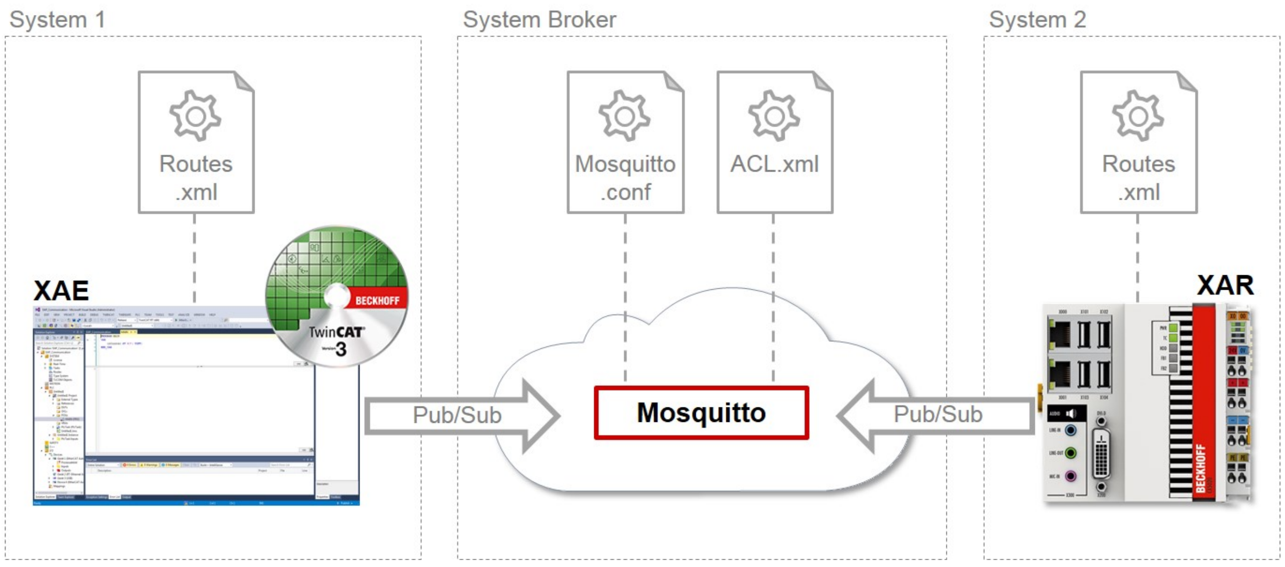
ADS connections can be blocked by the firewall if this is used.


ADS over MQTT
Broker
ADS over MQTT
ADS Router
ADS
ADS
Customer ADS application
TwinCAT
TwinCAT 3

7 Example

This example merely represents a workflow for setting up a test environment. All parameters such as certificate validity periods, key lengths, etc. are to be set according to the real environment and application.

The mode of operation and configuration of ADS-over-MQTT will now be explained in more detail below on the basis of an example. In the example the Eclipse Mosquitto Broker is used as the broker together with OpenSSL for creating the certificates. The Mosquitto Message Broker is to implement the exchange of data between a TwinCAT XAE and a TwinCAT XAR. In order to secure the communication, the TLS encryption protocol is used in combination with X.509 certificates or in combination with PSK. The structure of the application example is shown schematically in the illustration below.



The configuration files listed in the illustration must be created and adapted accordingly in order to use ADS-over-MQTT. Two examples are presented below.

In the first, ADS-over-MQTT is configured with TLS and X.509 certificates [▶ 25] and in the second with TLS and PreSharedKeys (PSK) [▶ 32].

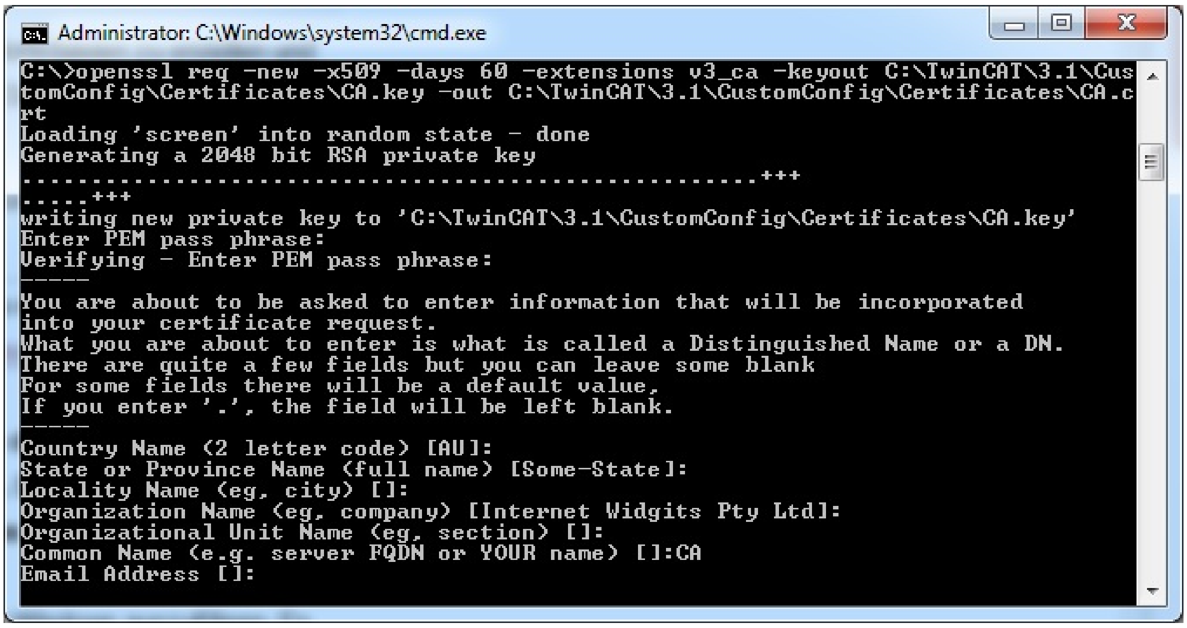
7.1 ADS-over-MQTT with TLS and X.509 certificates

In this section an example is introduced showing the configuration of ADS-over-MQTT with PSK and X.509 certificates. The individual steps to realize the communication interface are:

✓ TwinCAT 3.1 build 4022.0 or higher is installed on system 1 as the XAE version and on system 2 as the XAR version.

1. Generate the certificates for secure communication via TLS. To do this, use the program OpenSSL, which you can download from https://www.openssl.org/source/ and then install.
   Notice **With a Windows operating system the path to the OpenSSL configuration file must be set as an environment variable. Do this using the command line program of an x64 system with the following command:** `set OPENSSL_CONF=C:\OpenSSL-Win64\bin\openssl.cfg`
   On completion of the installation, execute the Windows command line program. The generation of the CA certificate (Certificate Authority) begins. The entry of a pass phrase is thereby demanded. Enter it and remember it and enter further information for the CA. The corresponding command for the generation of the CA certificate is:
   ```
   openssl req -new -x509 -days 60 -extensions v3_ca -keyout C:
   \TwinCAT\3.1\CustomConfig\Certificates\CA.key -out C:
   \TwinCAT\3.1\CustomConfig\Certificates\CA.crt
   ```

⇨ The result should look like this in the command line program:



Administrator: C:\Windows\system32\cmd.exe

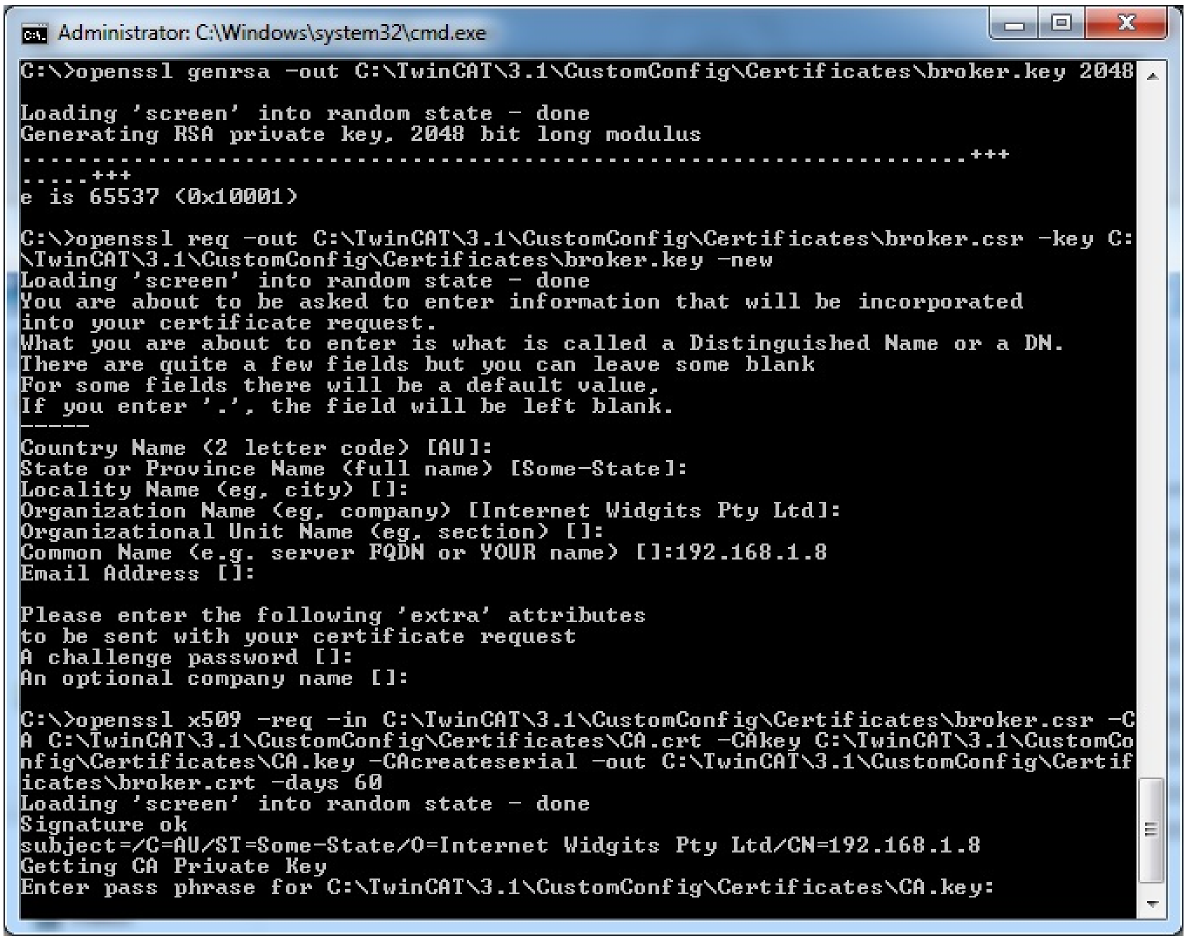
```
C:\>openssl req -new -x509 -days 60 -extensions v3_ca -keyout C:\TwinCAT\3.1\Cus
tomConfig\Certificates\CA.key -out C:\TwinCAT\3.1\CustomConfig\Certificates\CA.c
rt
Loading 'screen' into random state - done
Generating a 2048 bit RSA private key
.....................................................+++
.....+++
writing new private key to 'C:\TwinCAT\3.1\CustomConfig\Certificates\CA.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CA
Email Address []:
```

2. Generate the broker certificate. It is important here to use as the CN (Common Name) the host name or the IP address of the system on which the Mosquitto Message Broker is to be operated. Also, it must be ensured that the system is reachable via the IP address or the host name of the client. The following commands must be executed in the command line program to generate the broker certificates:
   Creating the certificate:
   `openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\broker.key 2048`

   Creating the Certificate Signing Request:
   `openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\broker.csr -key C:\TwinCAT\3.1\CustomConfig\Certificates\broker.key -new`

   Signing of the CSR by the previously created CA, for which the password is required that was specified when creating the CA:
   `openssl x509 -req -in C:\TwinCAT\3.1\CustomConfig\Certificates\broker.csr -CA C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:\TwinCAT\3.1\CustomConfig\Certificates\CA.key -CAcreateserial -out C:\TwinCAT\3.1\CustomConfig\Certificates\broker.crt -days 60`

⇨ The result should look like this in the command line program:



```
C:\>openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\broker.key 2048

Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
.......................................................................+++
.....+++
e is 65537 (0x10001)

C:\>openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\broker.csr -key C:
\TwinCAT\3.1\CustomConfig\Certificates\broker.key -new
Loading 'screen' into random state - done
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.1.8
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

C:\>openssl x509 -req -in C:\TwinCAT\3.1\CustomConfig\Certificates\broker.csr -C
A C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:\TwinCAT\3.1\CustomCo
nfig\Certificates\CA.key -CAcreateserial -out C:\TwinCAT\3.1\CustomConfig\Certif
icates\broker.crt -days 60
Loading 'screen' into random state - done
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=192.168.1.8
Getting CA Private Key
Enter pass phrase for C:\TwinCAT\3.1\CustomConfig\Certificates\CA.key:
```

3. Generate the two client certificates for the TwinCAT XAE and TwinCAT XAR. The OpenSSL commands for this are specified below.
   Generating the XAE certificate:
   ```
   openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.key
   2048
   ```

   Creating the CSR:
   ```
   openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.csr -
   key C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.key -new
   ```

   Signing of the CSR by the previously created CA, for which the password is required that was specified when creating the CA:
   ```
   openssl x509 -req -in C:
   \TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.csr -CA C:
   \TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:
   \TwinCAT\3.1\CustomConfig\Certificates\CA.key -CAcreateserial -out C:
   \TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.crt -days 60
   ```

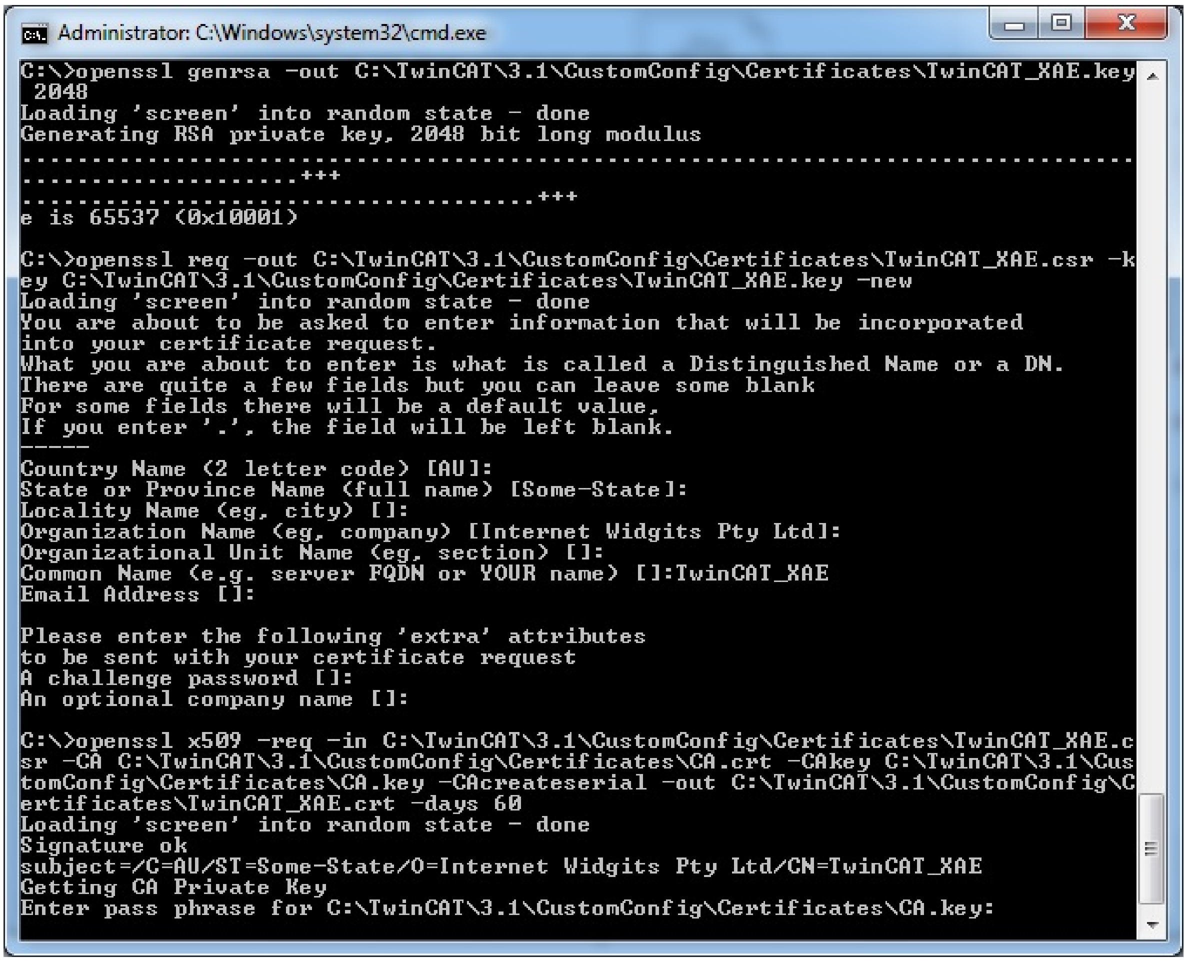
   Generating the XAR certificate:
   ```
   openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.key
   2048
   ```

   Creating the CSR:
   ```
   openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.csr -
   key C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.key -new
   ```

   Signing of the CSR by the previously created CA, for which the password is required that was specified when creating the CA:
   ```
   openssl x509 -req -in C:
   ```

```
\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.csr -CA C:
\TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:
\TwinCAT\3.1\CustomConfig\Certificates\CA.key -CAcreateserial -out C:
\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.crt -days 60
```

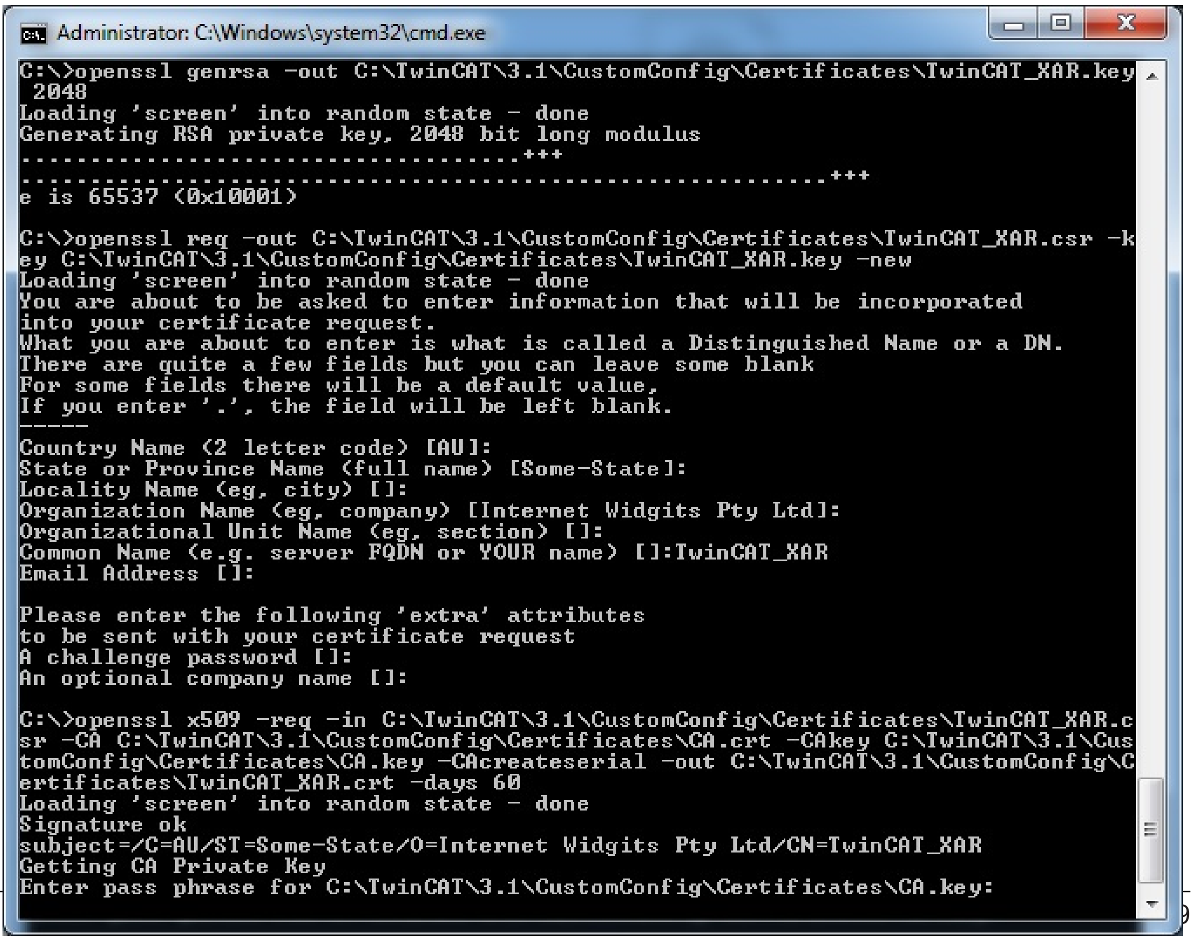
⇨ The result should look like this in the command line program:
TwinCAT XAE:



Administrator: C:\Windows\system32\cmd.exe

```
C:\>openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.key
 2048
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
................................................................................
....................+++
.......................................+++
e is 65537 (0x10001)

C:\>openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.csr -k
ey C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.key -new
Loading 'screen' into random state - done
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:TwinCAT_XAE
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

C:\>openssl x509 -req -in C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.c
sr -CA C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:\TwinCAT\3.1\Cus
tomConfig\Certificates\CA.key -CAcreateserial -out C:\TwinCAT\3.1\CustomConfig\C
ertificates\TwinCAT_XAE.crt -days 60
Loading 'screen' into random state - done
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=TwinCAT_XAE
Getting CA Private Key
Enter pass phrase for C:\TwinCAT\3.1\CustomConfig\Certificates\CA.key:
```

TwinCAT XAR:



Administrator: C:\Windows\system32\cmd.exe

```
C:\>openssl genrsa -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.key
 2048
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
.......................................+++
..........................................................+++
e is 65537 (0x10001)

C:\>openssl req -out C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.csr -k
ey C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.key -new
Loading 'screen' into random state - done
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:TwinCAT_XAR
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

C:\>openssl x509 -req -in C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAR.c
sr -CA C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt -CAkey C:\TwinCAT\3.1\Cus
tomConfig\Certificates\CA.key -CAcreateserial -out C:\TwinCAT\3.1\CustomConfig\C
ertificates\TwinCAT_XAR.crt -days 60
Loading 'screen' into random state - done
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=TwinCAT_XAR
Getting CA Private Key
Enter pass phrase for C:\TwinCAT\3.1\CustomConfig\Certificates\CA.key:
```

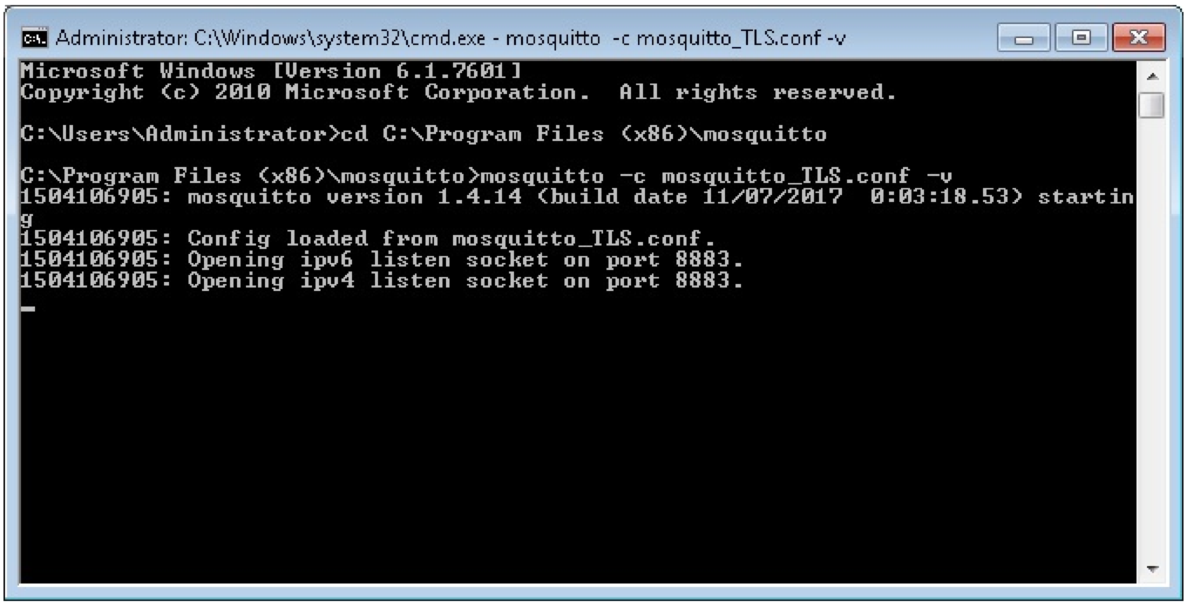
4. Install the Mosquitto Broker after generating the certificates. Download it from https://mosquitto.org/download/ and install it on the appropriate system.
5. Following the installation of the Mosquitto Broker, create the configuration file mosquitto_TLS.conf for it for the use of TLS with certificates. Choose the Mosquitto installation folder (default: C:\Program Files (x86)\mosquitto) as the storage location. The configuration file should contain the following entries:

```
port 8883
allow_anonymous false
require_certificate true
use_identity_as_username true
cafile C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt
certfile C:\TwinCAT\3.1\CustomConfig\Certificates\broker.crt
keyfile C:\TwinCAT\3.1\CustomConfig\Certificates\broker.key
auth_plugin C:\TwinCAT\AdsApi\TcMqttPlugin\TcMqttPlugin.dll
auth_opt_xml_file C:\TwinCAT\AdsApi\TcMqttPlugin\ACL.xml
```

6. Now start the Mosquitto Message Broker via the Windows command line program. To do this, switch to the Mosquitto installation directory and execute the command listed below. With this command, -v causes the output of the messages that are sent or rejected by the broker. This option is particularly useful during tests.

```
mosquitto -c mosquitto_TLS.conf -v
```

⇨ The subsequent result should look like this:


Administrator: C:\Windows\system32\cmd.exe - mosquitto -c mosquitto_TLS.conf -v

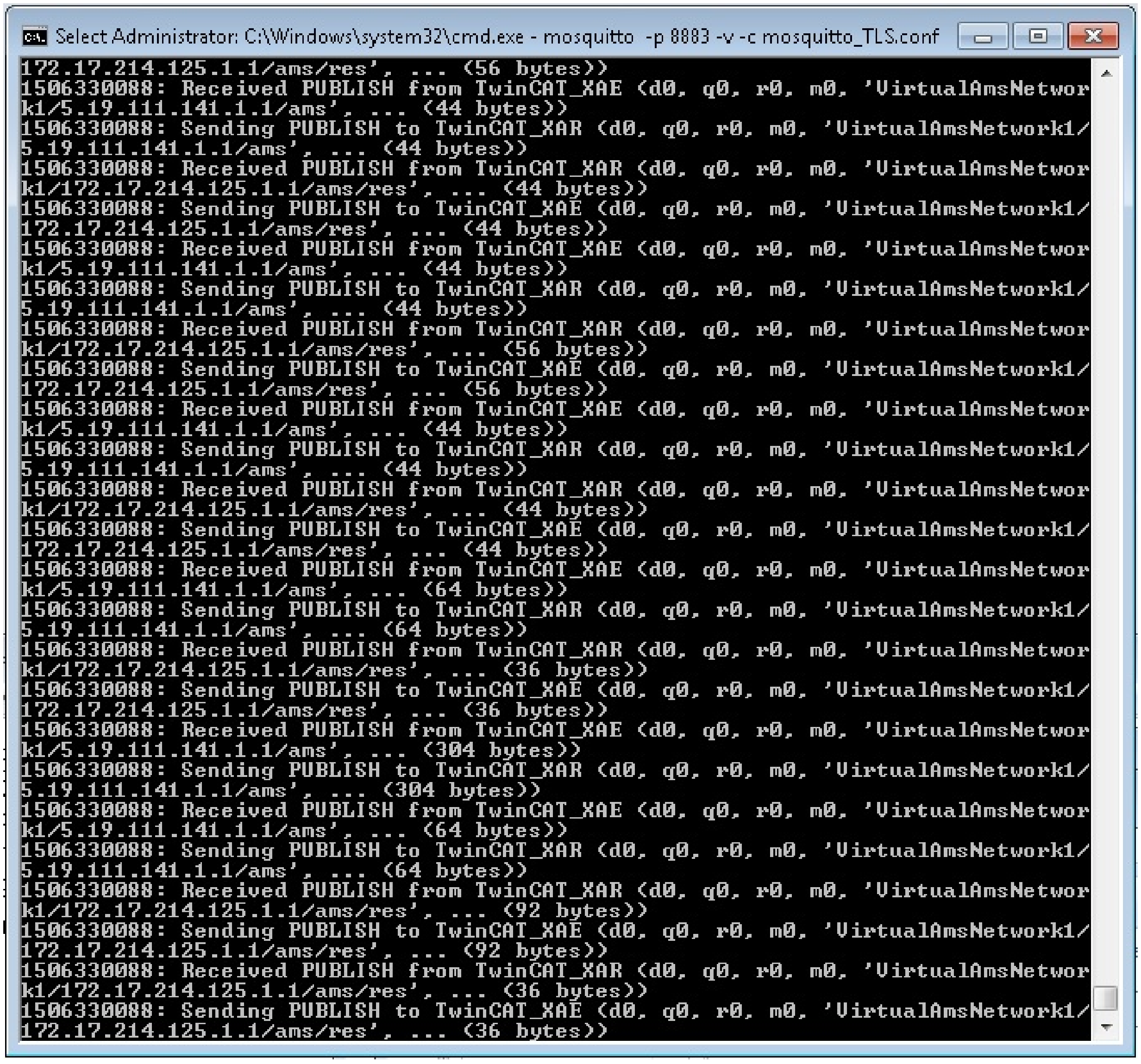
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2010 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd C:\Program Files (x86)\mosquitto

C:\Program Files (x86)\mosquitto>mosquitto -c mosquitto_TLS.conf -v
1504106905: mosquitto version 1.4.14 (build date 11/07/2017  0:03:18.53) startin
g
1504106905: Config loaded from mosquitto_TLS.conf.
1504106905: Opening ipv6 listen socket on port 8883.
1504106905: Opening ipv4 listen socket on port 8883.
_
```

7. Next, create the ACL.xml for the Mosquitto in which the access rights of the clients are defined. Store it in the directory C:\TwinCAT\AdsApi\TcMqttPlugin\. Make the following entries in the ACL.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcMqttAclConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:
\TwinCAT\3.1\Config\Modules\TcMqttAclConfig.xsd">
<Ams>
    <Topic>VirtualAmsNetwork1</Topic>
    <User>
        <Name>TwinCAT_XAE</Name>
    </User>
    <User>
        <Name>TwinCAT_XAR</Name>
        <Access>TwinCAT_XAE</Access>
    </User>
</Ams>
</TcMqttAclConfig>
```

8. Now configure the TwinCAT XAE and TwinCAT XAR for ADS-over-MQTT. To do this, create a folder with the name "Routes" on both systems in the directory C:\TwinCAT\3.x\Target\ in which you then generate a file with the name "MyRoute.xml" (the file name is arbitrary). The contents of the file from the TwinCAT XAE are shown below. Adapt the paths for the TwinCAT XAR in the <Cert> and <Key> fields accordingly. It is important that the same entry is always made in the <Address> field as for the CN of the Mosquitto Broker certificate.
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/
TcConfig">
<RemoteConnections>
    <Mqtt>
        <Address Port="8883">192.168.1.8</Address>
        <Topic>VirtualAmsNetwork1</Topic>
        <Tls>
            <Ca>C:\TwinCAT\3.1\CustomConfig\Certificates\CA.crt</Ca>
            <Cert>C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.crt</
Cert>
            <Key>C:\TwinCAT\3.1\CustomConfig\Certificates\TwinCAT_XAE.key</
Key>
        </Tls>
    </Mqtt>
</RemoteConnections>
</TcConfig>
```
9. Re-initialize the TwinCAT router in each case so that the stored configuration of ADS-over-MQTT becomes effective for the TwinCAT systems. This is done by switching from RUN mode to CONFIG mode or from CONFIG mode to CONFIG mode again.

⇨ Finally, check whether a connection can be established from the XAE to the XAR. If so, the outputs of the Mosquitto Message Broker should look like this:


Select Administrator: C:\Windows\system32\cmd.exe - mosquitto -p 8883 -v -c mosquitto_TLS.conf

```
172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (304 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (304 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (92 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (92 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (36 bytes))
```

⇨ ADS-over-MQTT with certificate-based TLS has thus been successfully set up for TwinCAT XAE and XAR.

7.2 ADS-over-MQTT with TLS and PSK

Apart from the use of TLS with certificates, MQTT-over-ADS can also be configured on the basis of PSK (Pre Shared Key). A short example will also be introduced for this application, which will support you in the implementation. The following steps have to be carried out:

1. First of all, create the Mosquitto configuration file (mosquitto_PSK.conf) in the Mosquitto installation folder (default: C:\Program Files (x86)\mosquitto). Then make the following entries in the file:
   ```
   auth_plugin C:\TwinCAT\AdsApi\TcMqttPlugin\TcMqttPlugin.dll
   auth_opt_xml_file C:\TwinCAT\AdsApi\TcMqttPlugin\ACL.xml
   port 8883
   psk_hint something
   use_identity_as_username true
   ```
2. In the next step, run the Mosquitto Message Broker. The command for this is:
   ```
   mosquitto -c mosquitto_PSK.conf -v
   ```
3. Enter the key for the TwinCAT XAR and XAE in the ACL.xml:
   ```
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <TcMqttAclConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ```

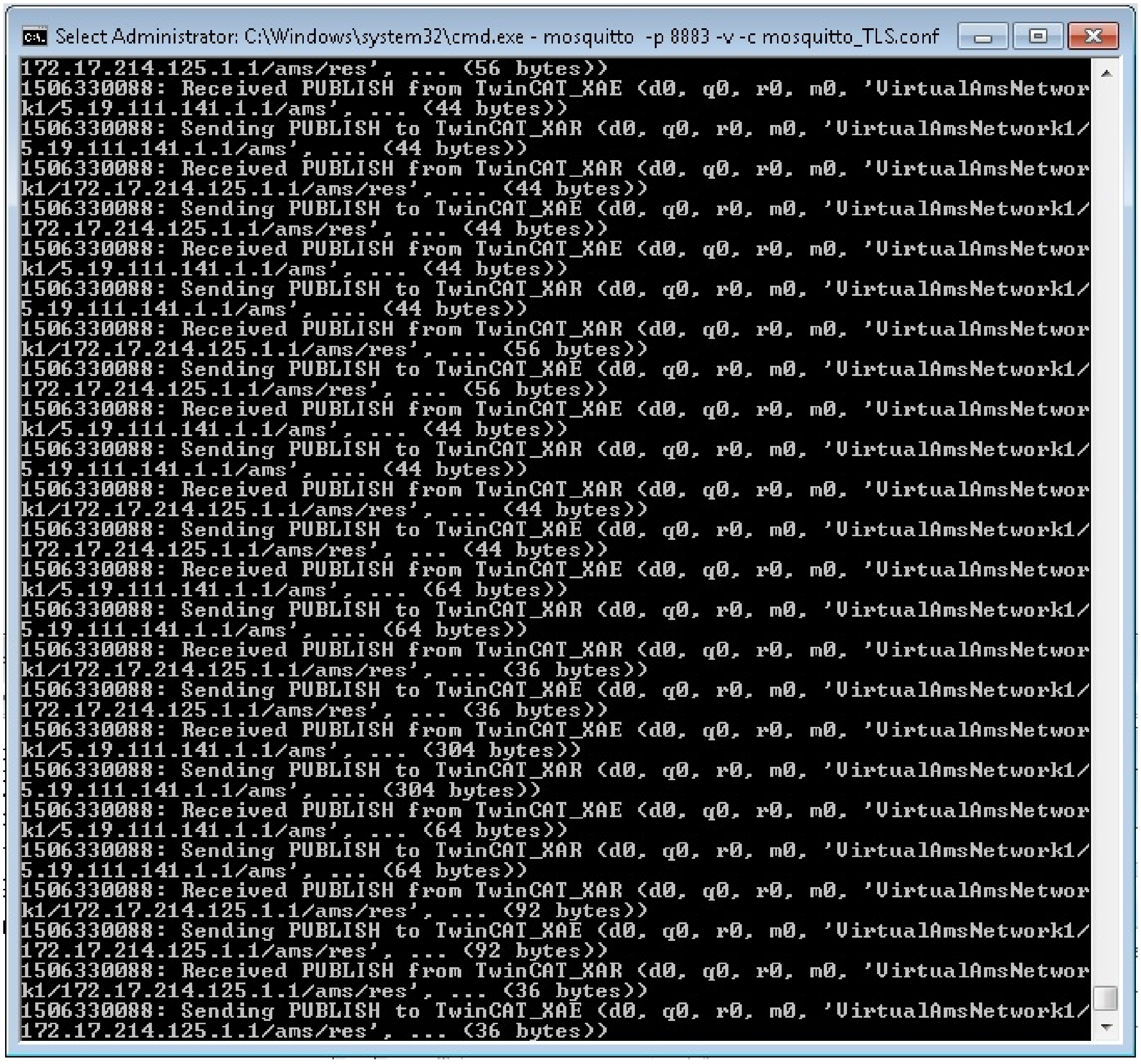
```
xsi:noNamespaceSchemaLocation="C:
\TwinCAT\3.1\Config\Modules\TcMqttAclConfig.xsd">
<Psk>
    <Identity>TwinCAT_XAE</Identity>
    <Pwd>abcdef1234!</Pwd>
</Psk>
<Psk>
    <Identity>TwinCAT_XAR</Identity>
    <Pwd>ghijkl5678?</Pwd>
</Psk>
<Ams>
    <Topic>VirtualAmsNetwork1</Topic>
   <User>
       <Name>TwinCAT_XAE</Name>
   </User>
    <User>
       <Name>TwinCAT_XAR</Name>
       <Access>TwinCAT_XAE</Access>
   </User>
</Ams>
</TcMqttAclConfig>
```

4. Also announce the key defined in the ACL.xml to the TwinCAT XAR and XAE. To do this, adapt or create the Routes.xml in the folder C:\TwinCAT\3.x\Target\Routes on both systems. The entries for the TwinCAT XAE are listed below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TcConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.beckhoff.com/schemas/2015/12/
TcConfig">
<RemoteConnections>
    <Mqtt>
        <Address Port="8883">192.168.1.8</Address>
        <Topic>VirtualAmsNetwork1</Topic>
        <Psk>
            <Identity>TwinCAT_XAE</Identity>
            <Pwd>abcdef1234!</Pwd>
        </Psk>
    </Mqtt>
</RemoteConnections>
</TcConfig>
```

5. The entries for the TwinCAT XAR are almost identical. You only need to adapt the values of the fields <Identity> and <Pwd> according to the details in the ACL.xml.
6. Once the configuration of Routes.xml on both systems is complete, reinitialize each TwinCAT router. To do this, switch from RUN mode to CONFIG mode or from CONFIG mode to CONFIG mode again.

7. Then check on the basis of the outputs of the Mosquitto Message Broker whether both systems can connect to the broker:



Select Administrator: C:\Windows\system32\cmd.exe - mosquitto -p 8883 -v -c mosquitto_TLS.conf

```
172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (56 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (44 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (304 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (304 bytes))
1506330088: Received PUBLISH from TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwork1/
5.19.111.141.1.1/ams', ... (64 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (92 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (92 bytes))
1506330088: Received PUBLISH from TwinCAT_XAR (d0, q0, r0, m0, 'VirtualAmsNetwor
k1/172.17.214.125.1.1/ams/res', ... (36 bytes))
1506330088: Sending PUBLISH to TwinCAT_XAE (d0, q0, r0, m0, 'VirtualAmsNetwork1/
172.17.214.125.1.1/ams/res', ... (36 bytes))
```

⇨ ADS-over-MQTT with PSK-based TLS has thus been successfully set up for TwinCAT XAE and XAR.

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com