MQTT

straton user guide – Rev. 7

sales@straton-plc.com

CP_R31_006_H

straton

Content

1. Overview

The goal of this tutorial is to teach you how to use MQTT communication protocol. To attain this goal, we will see two possibilities.

One with a MQTT broker installed on a raspberry, with another raspberry communicating with it.

The other with an online MQTT broker which will need only one raspberry.

We will firstly see how to communicate without certification and then integrate the certification process.

This tutorial will guide you through the configuration and use of MQTT in straton applications.

straton supports MQTT V3.1 or V3.1.1 over TCP.

2. Requirement and setup

Requires straton 9.1 Editor and Runtime or higher.

3. Setup and install with the MQTT Broker

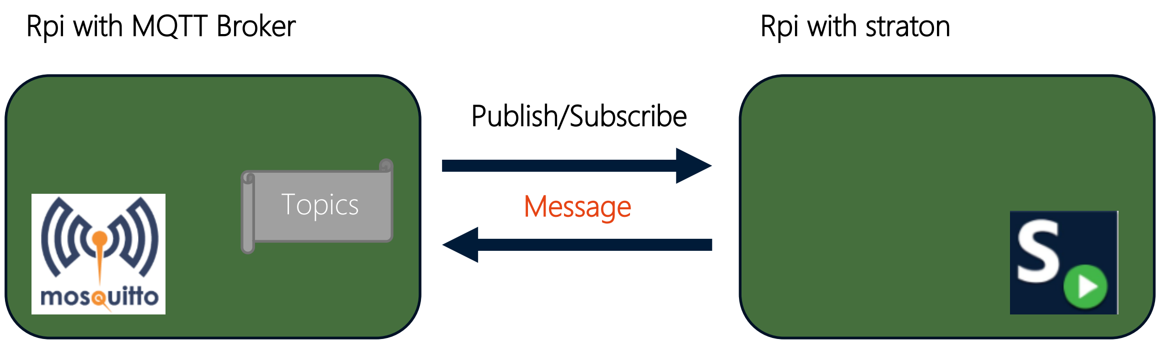
3.1. Prerequisites

For this demo application we will at first use two Raspberry Pi 4 communicating between each other in MQTT. Then, for encrypted exchanges we will replace the Raspberry hosting the straton Runtime by a Windows MQTT Client: MQTTBox.

One Raspberry will run a Mosquitto broker and subscribe to messages coming from the second one.

The second one will run the straton Runtime for Raspberry which can be downloaded from our website:

https://straton-plc.com/telechargements/



3.2. Setup the straton Runtime for Raspberry (Rpi #1)

The straton runtime must be installed on one of the Raspberry. Please follow the tutorial accessible from the delivery and also from our website > HERE <.

If not already done, make the Runtime executable

```
# sudo chmod +x t5runtimeName
```

Then start it using the super-user mode:

```
# sudo ./t5runtimeName
```

3.3. Setup the Mosquitto broker (Rpi #2)

The Mosquitto broker must be installed on a second Raspberry.

First, search for available updates for your system:

```
# sudo apt update
```

Then install the Mosquitto broker:

```
# sudo apt install -y mosquitto mosquitto-clients
```

Once the installation is finished, enable the broker to start on boot

```
# sudo systemctl enable mosquitto.service
```

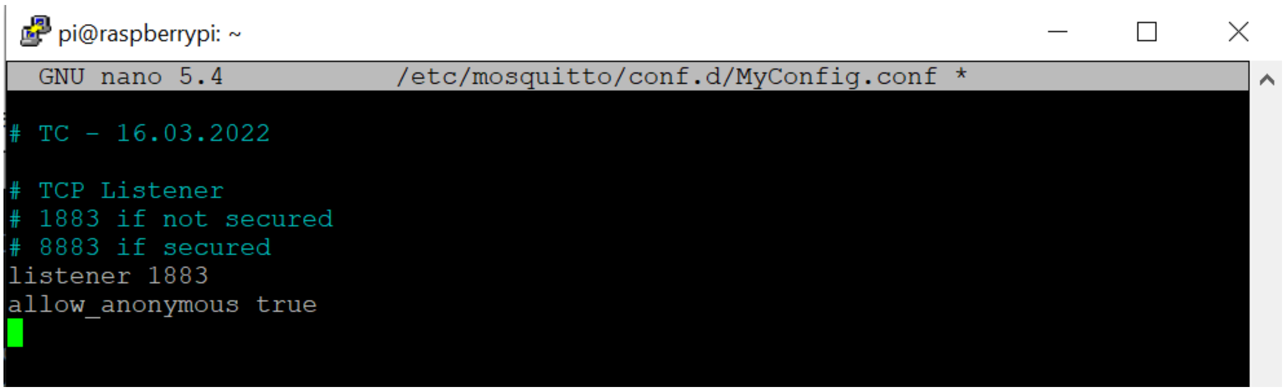
Check that it is properly installed (note that this could indicate it is already launched and using port 1883):

```
# mosquitto -v
```

NOTE: since the version 2.0 of mosquitto it is needed to run it with the parameter "-c" to load a configuration file.

Create a configuration file which looks like the following one

```
# sudo nano /etc/mosquitto/conf.d/MyConfig.conf
```
(<= for example)



```
  GNU nano 5.4              /etc/mosquitto/conf.d/MyConfig.conf *

# TC - 16.03.2022

# TCP Listener
# 1883 if not secured
# 8883 if secured
listener 1883
allow_anonymous true
```

Run mosquitto by loading the configuration

```
# mosquitto -c /etc/mosquitto/conf.d/MyConfig.conf -v
```

If not already launched (it can be checked with a "`top`" command), the "`-v`" option allows to start it in verbose mode, allowing to see debug messages. To run the Mosquitto broker in background use:

```
    # mosquitto -d
```

4. Configuration in Straton with the MQTT Broker

4.1. Fieldbus Configurations

See 1010 below.

Open the IO Drivers window using the tool bar ( ) or right click on the project > Insert Shortcut > Fieldbus configurations.

Insert a new Fieldbus using the tool bar ( ) or menu Insert > Insert configuration and select the "MQTT Client" driver.

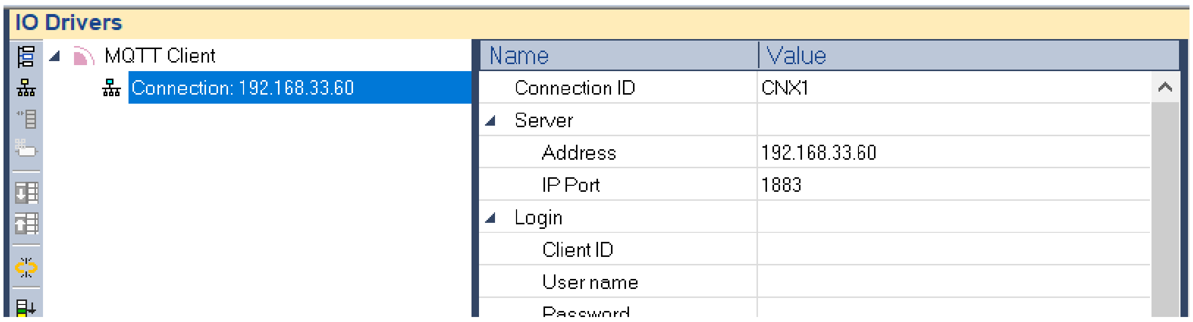
There are no parameters to set on the first level of the Configuration.

Insert a Connection ( ) or Menu > Insert > Insert Master/Port.

Here must be set:

- Connection ID = ID used by the MQTT blocks to know to which connection these are liked to
- Server/Address = the IP address of the Raspberry hosting the MQTT broker
- Server/IP Port = port used by the MQTT broker (by default this is 1883)

Other parameters should not be changed. Refer to the Editor's Online Help for more details.



Name	Value
Connection ID	CNX1
Server	
Address	192.168.33.60
IP Port	1883
Login	
Client ID	
User name	
Password	

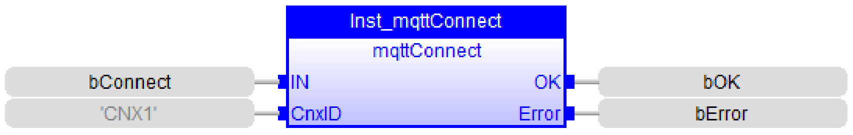
4.2. Programs and MQTT blocks

Configuring the MQTT exchanges is made through function blocks.

Firstly, create a new instance of MQTTCONNECT block. As a first input 'IN", add a BOOL variable which will allow to establish or not the connection.

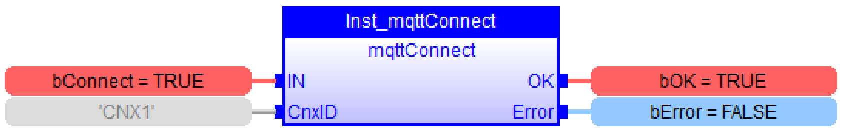
As connection ID “CnxID”, put the same as in the Fieldbus Configurations.

Select the block and press F1 for more information.



Once this very small program is ready, build the application and download it on the Raspberry hosting the straton Runtime.

Then check that the connection with the MQTT broker (which is on the other Raspberry) can be established putting the “IN” input to TRUE.



If the connection fails, try to set the “Clean session” option in the MQTT Fieldbus Configuration, then rebuild and download the updated application (it will remove pending messages to be published).

If this still fails, check that you set the right IP address and port and, as another option, try to kill the mosquitto service on Linux, then start it using “mosquitto -v” (it shows more errors than if it is running in background using the -d option).

5. Exchanging messages with the MQTT Broker

5.1. Launching the MQTT Broker on the Raspberry

In the Raspberry hosting the Mosquitto broker, check if it is already running or not

```
# ps -A | grep mosq
```

If the previous command shews nothing, the broker is not running. Start it in background

```
# mosquitto -d
```

Then subscribe to a topic:

```
# mosquitto_sub -d -t MyTopic
```

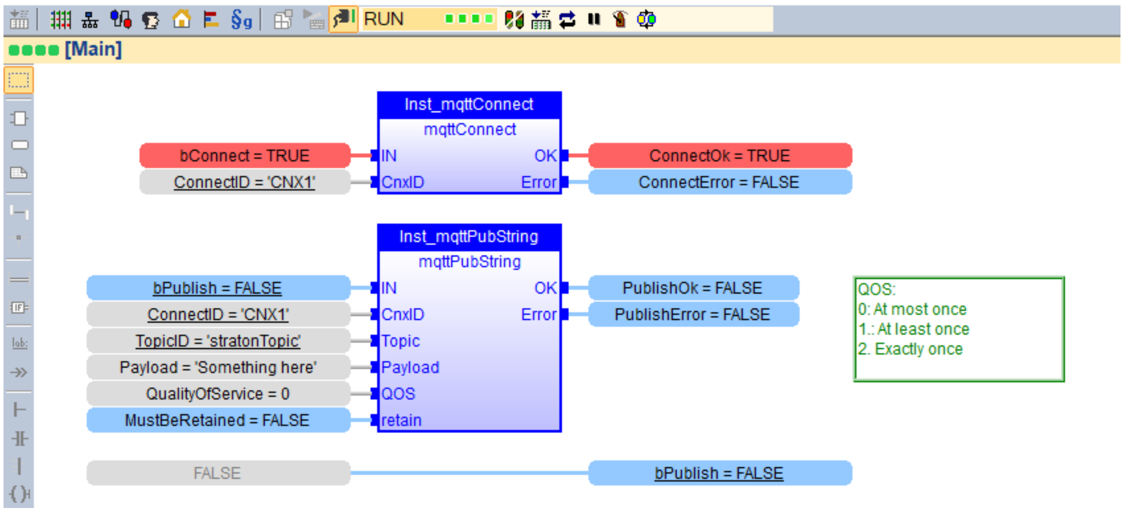
5.2. Publisher

In the straton project, declare an instance of “MQTTPUBSTRING” block and configure it to:

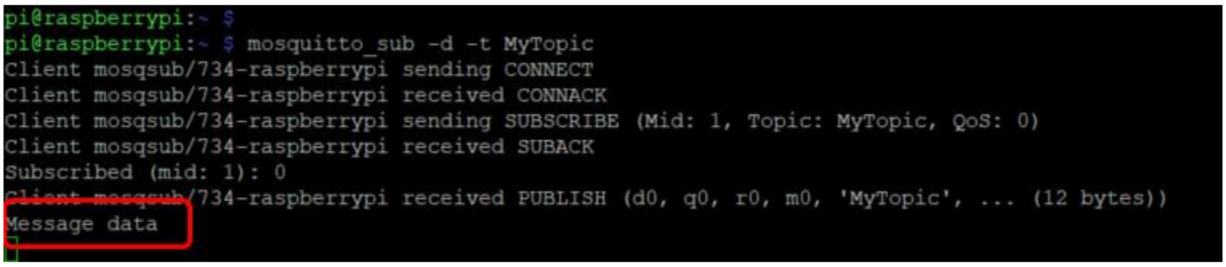
- Connect to the broker via its Fieldbus Configurations’ connection ID
- Publish to the topic “MyTopic”
- Publish a message you can choose (eg. ‘Message Data’)

Then build the application and go online.

Setting the “IN” input of the MQTTPUBSTRING block to TRUE will send/publish the message (Payload) to the MQTT topic.



In the Raspberry hosting the broker, in which you subscribed to messages from MyTopic, you must see the message sent by the straton MQTT Client.



```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ mosquitto_sub -d -t MyTopic
Client mosqsub/734-raspberrypi sending CONNECT
Client mosqsub/734-raspberrypi received CONNACK
Client mosqsub/734-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: MyTopic, QoS: 0)
Client mosqsub/734-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/734-raspberrypi received PUBLISH (d0, q0, r0, m0, 'MyTopic', ... (12 bytes))
Message data
```

5.3. Subscriber

You can add the subscribe code on the same Raspberry if you want to test it.

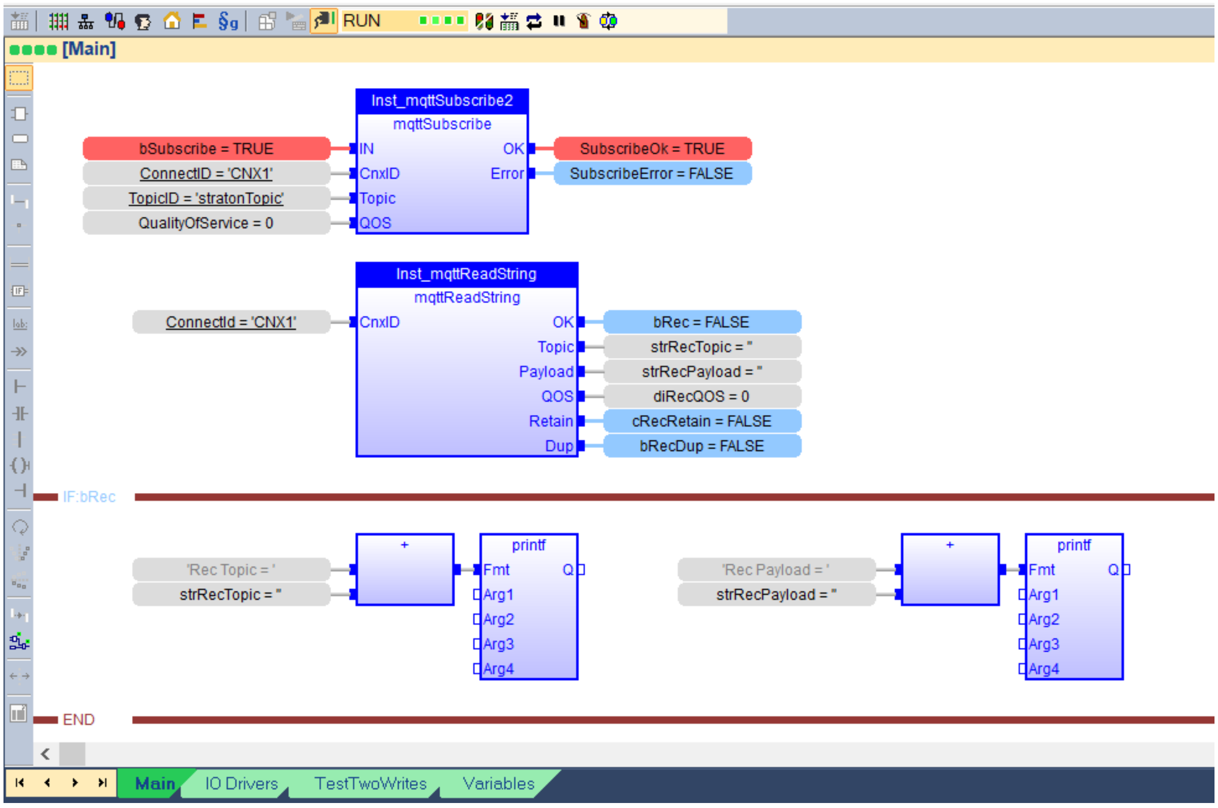
In the Raspberry in which you want to subscribe to messages from MyTopic, you must declare the same Fieldbus Configuration, same MQTTCONNECT block, and declare an instance of “MQTTSUBSCRIBE” to:

- Connect to the broker via its Fieldbus Configurations’ connection ID

- Subscribe to the topic "MyTopic"

Then use MQTTREADSTRING and, because of the outputs being updated on each cycle, PRINTF blocks to display the received messages.

Once the MQTTPUBSTRING.IN input is set to TRUE in the Publisher's application, to send the message, then the Subscriber's application displays the received payload:



6. Securing data exchanges with the MQTT Broker

6.1. Configuring SSL on the Mosquitto broker

We will use self-signed certificates, created with openssl. "Self-signed certificates" means that the Certification Authority (CA) will by ourselves. To check if openssl is installed, you can run "`openssl help`". It not installed, install it using

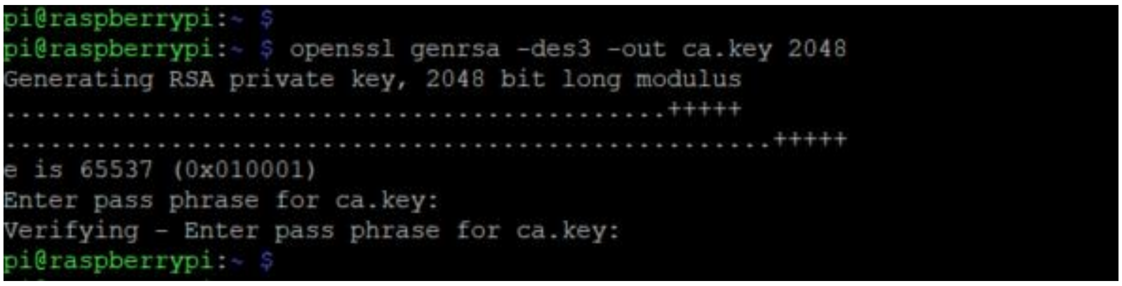
```
# sudo apt-get install openssl
```

6.1.1. Create a key pair for the CA

Run the following command

```
# openssl genrsa -des3 -out ca.key 2048
```

(remember or note the passphrase for ca.key somewhere)

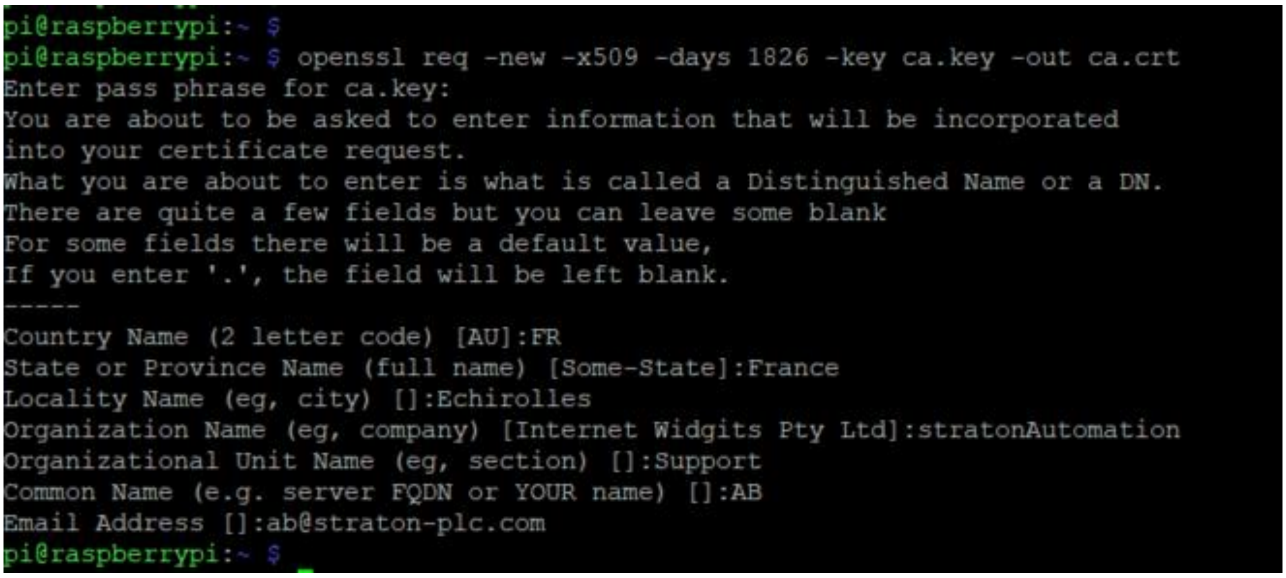


```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ openssl genrsa -des3 -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
............................................+++++
...................................................+++++
e is 65537 (0x010001)
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:
pi@raspberrypi:~ $
```

6.1.2. Create ca.crt certificate for the CA using the ca.key

Run the following command

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

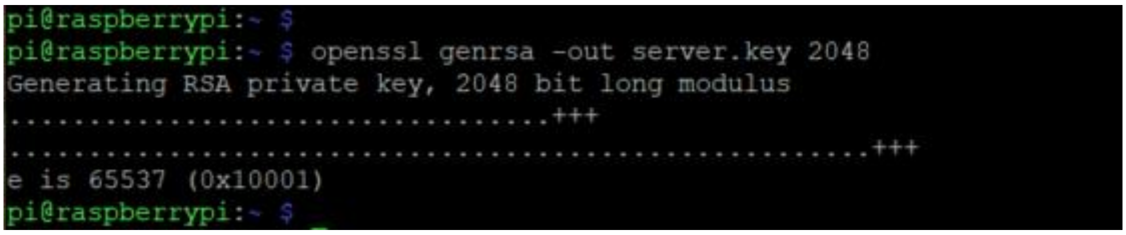


```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
Enter pass phrase for ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Echirolles
Organization Name (eg, company) [Internet Widgits Pty Ltd]:stratonAutomation
Organizational Unit Name (eg, section) []:Support
Common Name (e.g. server FQDN or YOUR name) []:AB
Email Address []:ab@straton-plc.com
pi@raspberrypi:~ $
```

6.1.3. Create Server key pair which will be used by the broker

Run the following command

```
# openssl genrsa -out server.key 2048
```



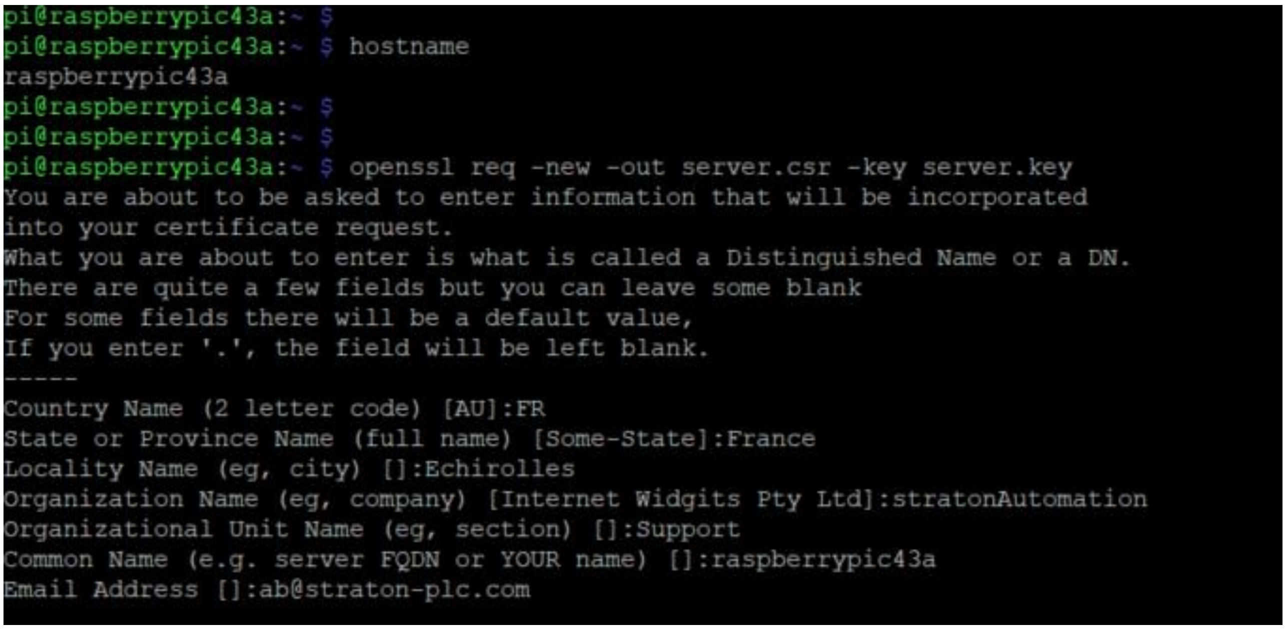
```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.................................+++
...................................................+++
e is 65537 (0x10001)
pi@raspberrypi:~ $
```

6.1.4. Create certificate request using the server key

When filling out the information which will be put into the certificate, the common name is an important field. It is usually the domain name of the server. As for this demo we are doing a local application, it can also be the device's hostname or the IP address or even full domain name. Just think to note it because it could be needed for the client connection.

Run the following command

```
# openssl req -new -out server.csr -key server.key
```



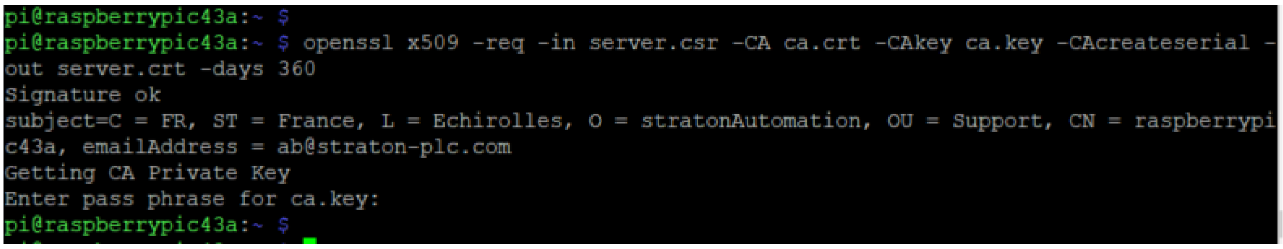
```
pi@raspberrypic43a:~ $
pi@raspberrypic43a:~ $ hostname
raspberrypic43a
pi@raspberrypic43a:~ $
pi@raspberrypic43a:~ $
pi@raspberrypic43a:~ $ openssl req -new -out server.csr -key server.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Echirolles
Organization Name (eg, company) [Internet Widgits Pty Ltd]:stratonAutomation
Organizational Unit Name (eg, section) []:Support
Common Name (e.g. server FQDN or YOUR name) []:raspberrypic43a
Email Address []:ab@straton-plc.com
```

This does not has to be sent to a Certification Authority as here we are the CA.

6.1.5. Use the ca.key to verify and sign the server certificate

Run the following command

```
      # openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out server.crt -days 360
```



```
pi@raspberrypic43a:~ $
pi@raspberrypic43a:~ $ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -
out server.crt -days 360
Signature ok
subject=C = FR, ST = France, L = Echirolles, O = stratonAutomation, OU = Support, CN = raspberrypi
c43a, emailAddress = ab@straton-plc.com
Getting CA Private Key
Enter pass phrase for ca.key:
pi@raspberrypic43a:~ $
```

6.1.6. Store the certificates and keys in Mosquitto

The last steps allowed to create 6 files. You can see it using the "`ls`" command. These are `ca.crt`, `ca.key`, `ca.srl`, `server.crt`, `server.csr` and `server.key`.

In the Mosquitto broker for Raspberry, certificates and keys are stored in the /etc/mosquitto folder. Here we need:

The CA certificate:	`ca.crt`
The Server certificate file:	`server.crt`
The Server private key:	`server.key`

To copy it in the right folder, use:

```
# sudo mv ca.crt /etc/mosquitto/ca_certificates/
# sudo mv server.crt /etc/mosquitto/certs
# sudo mv server.key /etc/mosquitto/certs
```

6.1.7. Configure the Mosquitto broker to handle secured communication

By default the broker starts on port 1883 and uses standard/tcp communication. We then have to configure it to handle the certificates we created and open the port which is usually used by MQTT for secured communication: 8883.

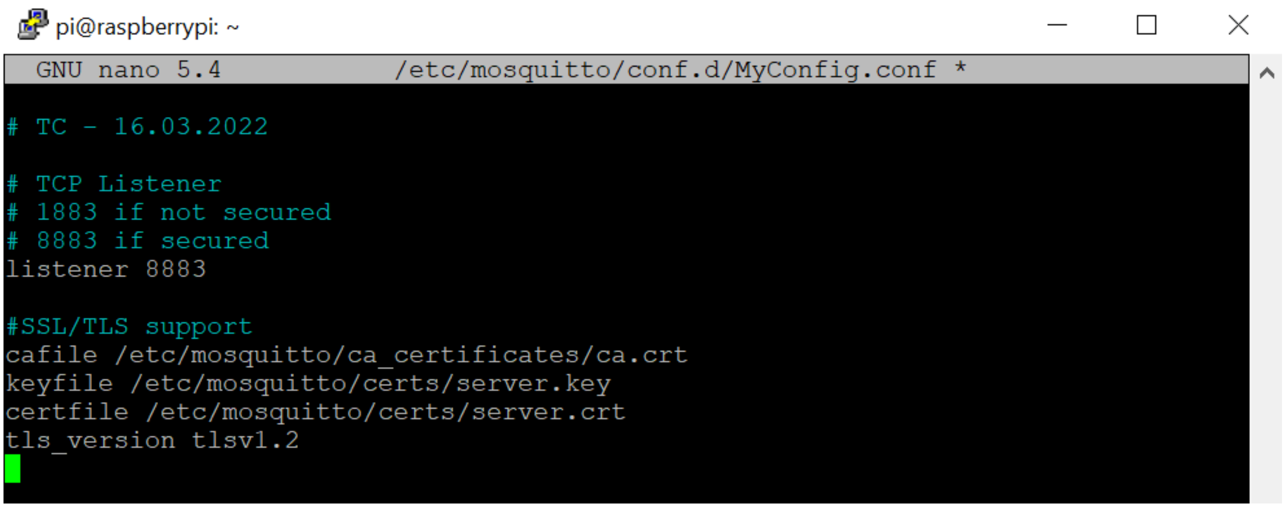
Normally, configuration files for Mosquitto are stored in the "`/etc/mosquitto/conf.d`" folder and have the "`*.conf`" extension.

Thus, start creating a new file there (or use the one you already created in step 4.2)

For example using:

```
# sudo nano /etc/mosquitto/conf.d/MyConfig.conf
```

Then, edit it to handle the certificates and keys we created and open port 8883. It should look like that:

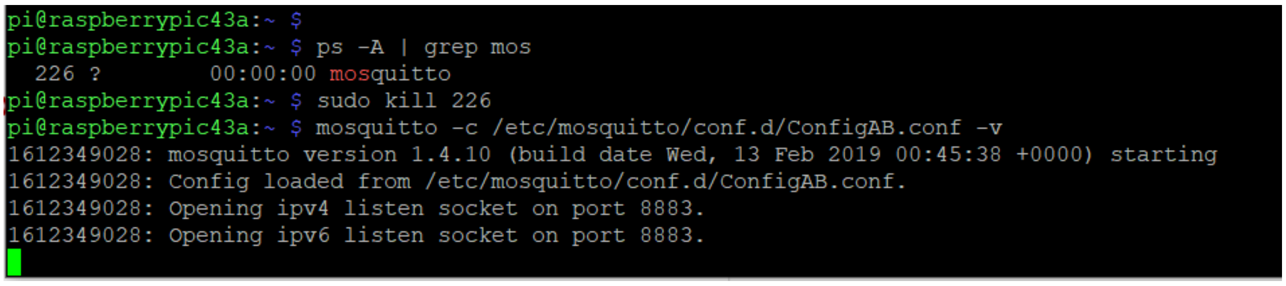


```
  GNU nano 5.4              /etc/mosquitto/conf.d/MyConfig.conf *

# TC - 16.03.2022

# TCP Listener
# 1883 if not secured
# 8883 if secured
listener 8883

#SSL/TLS support
cafile /etc/mosquitto/ca_certificates/ca.crt
keyfile /etc/mosquitto/certs/server.key
certfile /etc/mosquitto/certs/server.crt
tls_version tlsv1.2
```

Close and save the file using CTRL+X.

If the broker is running (`# ps -A | grep mosq`) kill it using its PID (`# sudo kill PID`) and restart it in verbose mode and be sure it takes in account the new configuration file:

```
# mosquitto -c /etc/mosquitto/conf.d/MyConfig.conf -v
```



```
pi@raspberrypic43a:~ $
pi@raspberrypic43a:~ $ ps -A | grep mos
  226 ?        00:00:00 mosquitto
pi@raspberrypic43a:~ $ sudo kill 226
pi@raspberrypic43a:~ $ mosquitto -c /etc/mosquitto/conf.d/ConfigAB.conf -v
1612349028: mosquitto version 1.4.10 (build date Wed, 13 Feb 2019 00:45:38 +0000) starting
1612349028: Config loaded from /etc/mosquitto/conf.d/ConfigAB.conf.
1612349028: Opening ipv4 listen socket on port 8883.
1612349028: Opening ipv6 listen socket on port 8883.
```

Then you we have to configure the Raspberry to communicate with the encrypted mode.

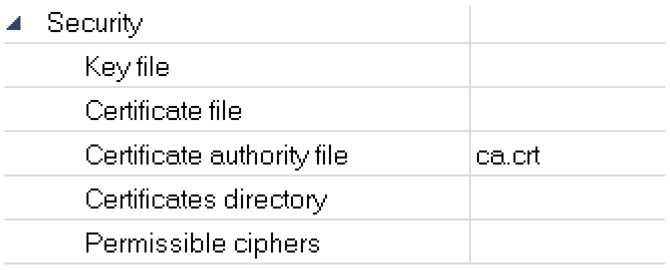
6.2. Fieldbus Configurations – Encrypted communication

The certificate you create before must be used on your application for encrypted communication. It's the ca.crt file

In this part, edit the Fieldbus Configurations in both application (Subscriber + Publisher) with:

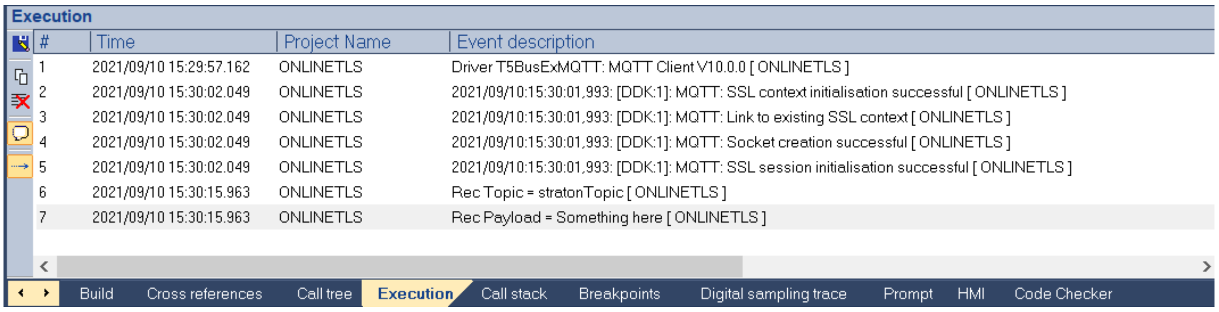
- IP Port = 8883
- Certificate authority file = /home/pi/ca.crt
- Certificate directory = /home/pi

Or in our case you can also do that if you put the .crt file in the directory of your application:



◢ Security	
Key file	
Certificate file	
Certificate authority file	ca.crt
Certificates directory	
Permissible ciphers	

Once the project is running, Subscriber can get data from the Publisher.

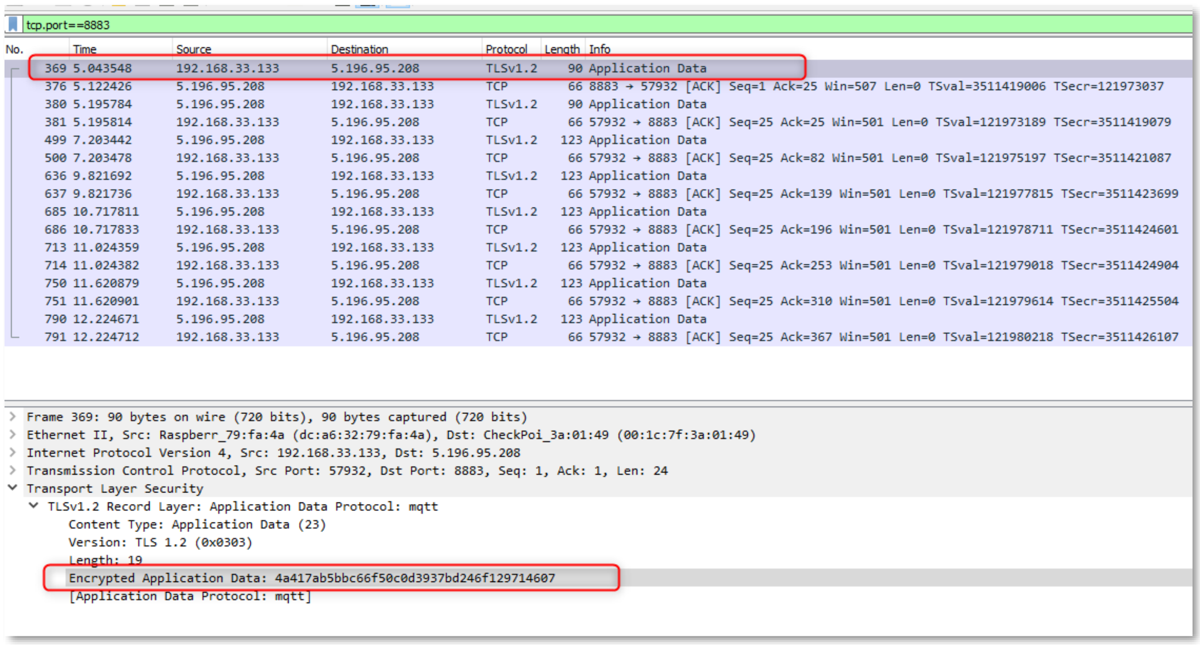


Execution

#	Time	Project Name	Event description
1	2021/09/10 15:29:57.162	ONLINETLS	Driver T5BusExMQTT: MQTT Client V10.0.0 [ONLINETLS]
2	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: SSL context initialisation successful [ONLINETLS]
3	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: Link to existing SSL context [ONLINETLS]
4	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: Socket creation successful [ONLINETLS]
5	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: SSL session initialisation successful [ONLINETLS]
6	2021/09/10 15:30:15.963	ONLINETLS	Rec Topic = stratonTopic [ONLINETLS]
7	2021/09/10 15:30:15.963	ONLINETLS	Rec Payload = Something here [ONLINETLS]

If you capture the frame between the two Raspberry you can see that the communication is encrypted.

See the FAQ.



```
tcp.port==8883

No.   Time       Source          Destination     Protocol Length Info
  369 5.043548   192.168.33.133  5.196.95.208    TLSv1.2    90 Application Data
  376 5.122426   5.196.95.208    192.168.33.133  TCP        66 8883 → 57932 [ACK] Seq=1 Ack=25 Win=507 Len=0 TSval=3511419006 TSecr=121973037
  380 5.195784   5.196.95.208    192.168.33.133  TLSv1.2    90 Application Data
  381 5.195814   192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=25 Win=501 Len=0 TSval=121973189 TSecr=3511419079
  499 7.203442   5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  500 7.203478   192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=82 Win=501 Len=0 TSval=121975197 TSecr=3511421087
  636 9.821692   5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  637 9.821736   192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=139 Win=501 Len=0 TSval=121977815 TSecr=3511423699
  685 10.717811  5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  686 10.717833  192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=196 Win=501 Len=0 TSval=121978711 TSecr=3511424601
  713 11.024359  5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  714 11.024382  192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=253 Win=501 Len=0 TSval=121979018 TSecr=3511424904
  750 11.620879  5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  751 11.620901  192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=310 Win=501 Len=0 TSval=121979614 TSecr=3511425504
  790 12.224671  5.196.95.208    192.168.33.133  TLSv1.2   123 Application Data
  791 12.224712  192.168.33.133  5.196.95.208    TCP        66 57932 → 8883 [ACK] Seq=25 Ack=367 Win=501 Len=0 TSval=121980218 TSecr=3511426107

> Frame 369: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
> Ethernet II, Src: Raspberr_79:fa:4a (dc:a6:32:79:fa:4a), Dst: CheckPoi_3a:01:49 (00:1c:7f:3a:01:49)
> Internet Protocol Version 4, Src: 192.168.33.133, Dst: 5.196.95.208
> Transmission Control Protocol, Src Port: 57932, Dst Port: 8883, Seq: 1, Ack: 1, Len: 24
v Transport Layer Security
  v TLSv1.2 Record Layer: Application Data Protocol: mqtt
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 19
      Encrypted Application Data: 4a417ab5bbc66f50c0d3937bd246f129714607
      [Application Data Protocol: mqtt]
```

7. Setup and install with the Online Broker

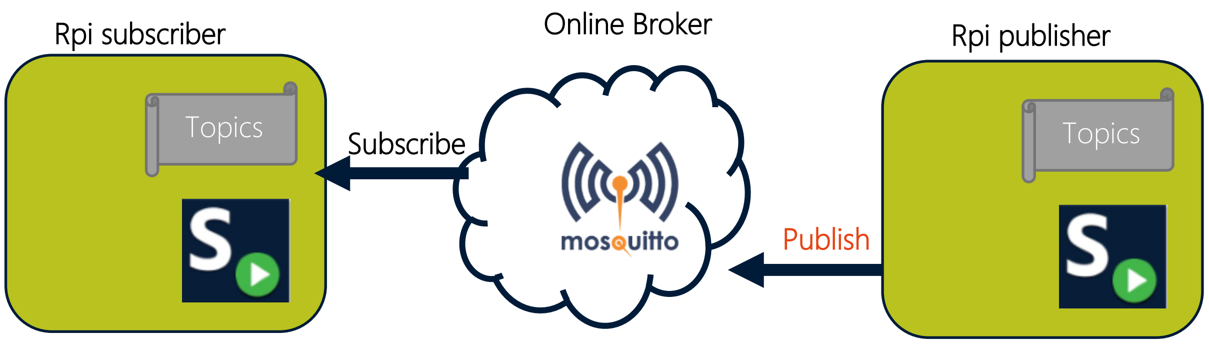
7.1. Prerequisites

For this demo application we will use two Raspberry Pi 4 communicating between each other over MQTT. Each Raspberry Pi 4 will run a straton Runtime. The MQTT broker will be the Mosquitto one, which is available directly online, without having to install anything.

One Rpi4 (publisher) will send data to the Broker.

The second one will subscribe to topics from the Online Broker.

IMPORTANT: This has been done to illustrate a complete application. If only one Raspberry is used, then you can do the Publish and Subscribe in the **same** application, on the same Raspberry. This usecase is described in part 5 of the present tutorial.



7.2. Setup the straton Runtime for Raspberry

The straton runtime must be installed on both Raspberry. Please follow the tutorial accessible from our website > HERE* <.

If not already done, make the Runtime executable

```
# sudo chmod +x t5runtimeName
```

Then start it using the super-user mode:

```
# sudo ./t5runtimeName
```

8. Configuration in straton with the Online Broker

8.1. Fieldbus Configurations – Using the Online broker without security

See 1010 below.

Open the IO Drivers window using the tool bar ( ) or right click on the project > Insert Shortcut > Fieldbus configurations.

Insert a new Fieldbus using the tool bar ( ) or menu Insert > Insert configuration and select the "MQTT Client" driver.

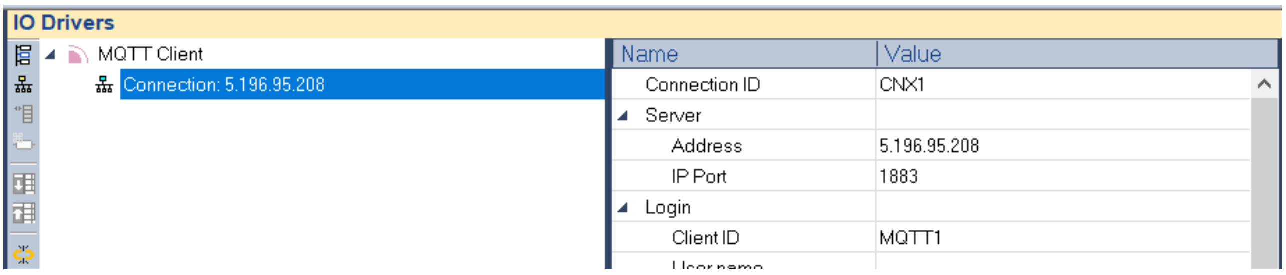
There are no parameters to set on the first level of the Configuration.

Insert a Connection ( ) or Menu > Insert > Insert Master/Port.

Here must be set:

- Connection ID = ID used as input by the MQTT blocks
- Server/Address = the IP address of the Mosquitto MQTT online broker (5.196.95.208 while writing these lines)
- Server/IP Port = port used by the MQTT broker (by default this is 1883)

Other parameters can be left with default values. Refer to the Editor's Online Help for more details.



Name	Value
Connection ID	CNX1
Server	
Address	5.196.95.208
IP Port	1883
Login	
Client ID	MQTT1

8.2. Programs and MQTT blocks

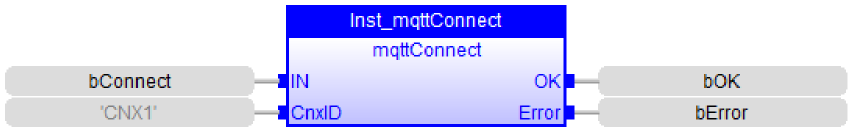
We do the same thing as previously, please go to the section Exchanging messages to look at the publisher and subscriber. Go here : 5 Exchanging messages

Configuring the MQTT exchanges is made through function blocks.

Firstly, create a new instance of "MQTTCONNECT" block. As a first input 'IN", add a BOOL variable which will allow to establish or not the connection.

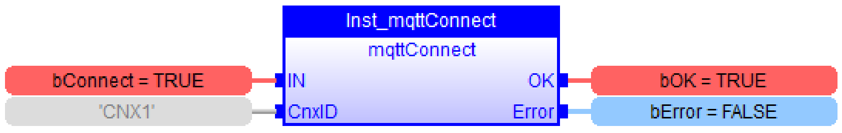
As connection ID "CnxID", put the same name as in the Fieldbus Configurations.

Select the block and press F1 for more information.



Once this very small program is ready, build the application and download it on the Raspberry hosting the straton Runtime.

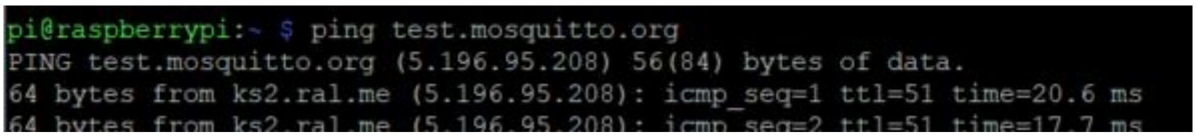
Then check that the connection with the online MQTT broker can be established using the block:



If the connection fails, try to set the "Clean session" option in the Fieldbus Configuration, then rebuild and download the updated application (it will remove pending messages to be published).

If this still fails, check that you set the right IP address and port in the Fieldbus Configuration. Also try to ping the online MQTT broker from the Raspberry to be sure it is accessible:

```
# ping test.mosquitto.org
```



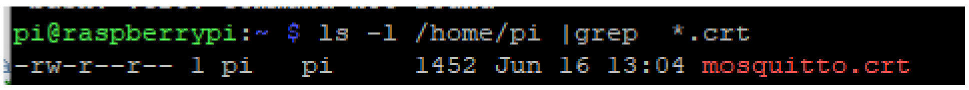
```
pi@raspberrypi:~ $ ping test.mosquitto.org
PING test.mosquitto.org (5.196.95.208) 56(84) bytes of data.
64 bytes from ks2.ral.me (5.196.95.208): icmp_seq=1 ttl=51 time=20.6 ms
64 bytes from ks2.ral.me (5.196.95.208): icmp_seq=2 ttl=51 time=17.7 ms
```

8.3. Configuring straton to use encrypted communications

Be careful, security is only implemented since Straton V10.0 build 2 (10503)

The next part of this document describes how to establish an encrypted communications between the Raspberry Pi 4 and the Mosquitto online broker. For this we need a CA (Certificate Authority) we can get from Mosquitto (https://test.mosquitto.org)

After downloading the certificate authority file we must put this file on both Rpi 4. (Eg. /home/pi/mosquito.crt). We can do that with the WinSCP tool or FileZilla.

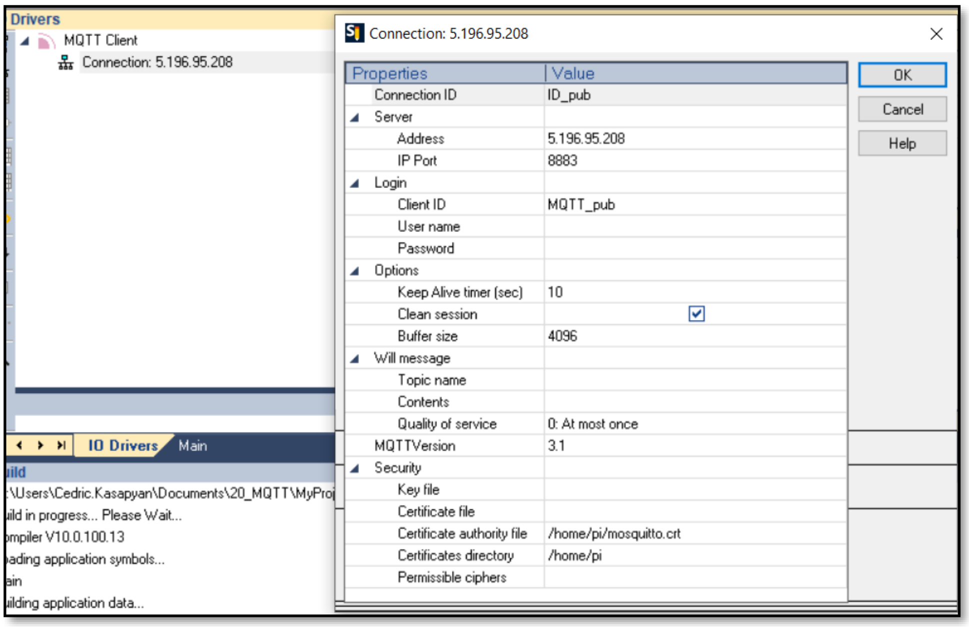


```
pi@raspberrypi:~ $ ls -l /home/pi |grep  *.crt
-rw-r--r-- 1 pi   pi      1452 Jun 16 13:04 mosquitto.crt
```

8.4. Fieldbus Configurations – Encrypted communication

In this part, edit the Fieldbus Configurations in both application (Subscriber + Publisher) with:

- IP Port = 8883
- Certificate authority file = /home/pi/mosquito.crt
- Certificate directory = /home/pi

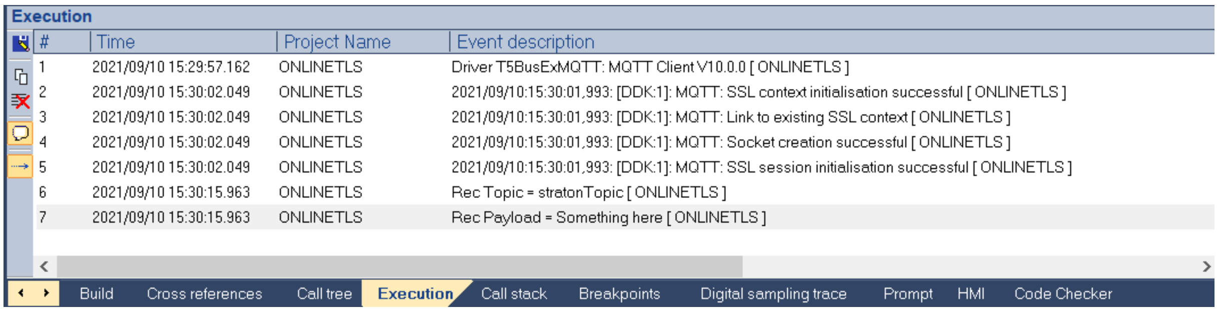


Connection: 5.196.95.208

Properties	Value
Connection ID	ID_pub
Server	
Address	5.196.95.208
IP Port	8883
Login	
Client ID	MQTT_pub
User name	
Password	
Options	
Keep Alive timer (sec)	10
Clean session	✓
Buffer size	4096
Will message	
Topic name	
Contents	
Quality of service	0: At most once
MQTTVersion	3.1
Security	
Key file	
Certificate file	
Certificate authority file	/home/pi/mosquitto.crt
Certificates directory	/home/pi
Permissible ciphers	

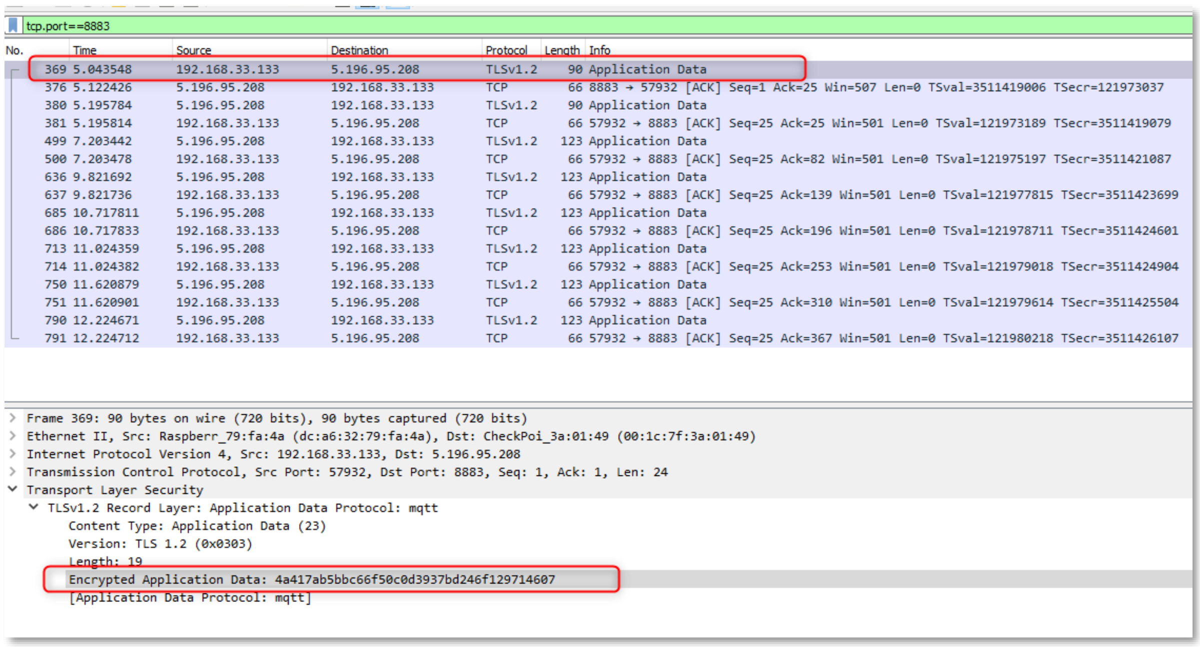
```
ild
:\Users\Cedric.Kasapyan\Documents\20_MQTT\MyProj
ild in progress... Please Wait...
mpiler V10.0.100.13
ading application symbols...
ain
ilding application data...
```

Once the project is running, Subscriber can get data from the Publisher.



#	Time	Project Name	Event description
1	2021/09/10 15:29:57.162	ONLINETLS	Driver T5BusExMQTT: MQTT Client V10.0.0 [ONLINETLS]
2	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: SSL context initialisation successful [ONLINETLS]
3	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: Link to existing SSL context [ONLINETLS]
4	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: Socket creation successful [ONLINETLS]
5	2021/09/10 15:30:02.049	ONLINETLS	2021/09/10:15:30:01,993: [DDK:1]: MQTT: SSL session initialisation successful [ONLINETLS]
6	2021/09/10 15:30:15.963	ONLINETLS	Rec Topic = stratonTopic [ONLINETLS]
7	2021/09/10 15:30:15.963	ONLINETLS	Rec Payload = Something here [ONLINETLS]

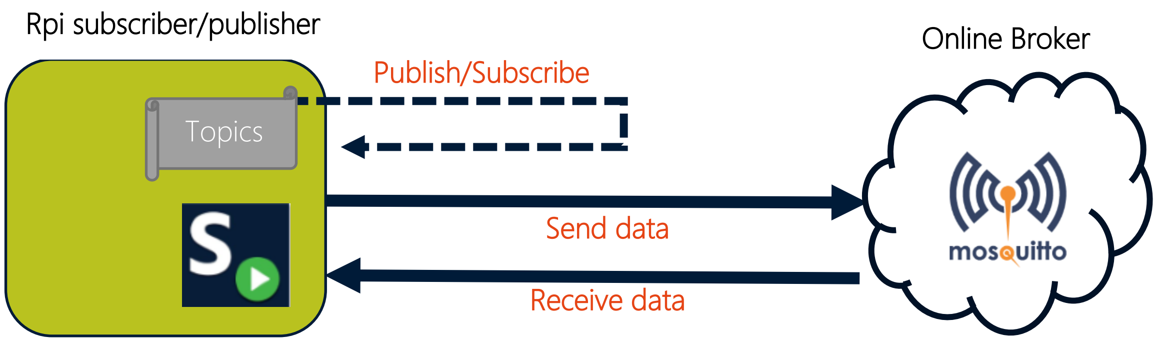
In Wireshark MQTT frames content is now fully encrypted:



```
tcp.port==8883
No.      Time         Source           Destination      Protocol  Length Info
    369 5.043548     192.168.33.133   5.196.95.208     TLSv1.2     90 Application Data
    376 5.122426     5.196.95.208     192.168.33.133   TCP         66 8883 → 57932 [ACK] Seq=1 Ack=25 Win=507 Len=0 TSval=3511419006 TSecr=121973037
    380 5.195784     5.196.95.208     192.168.33.133   TLSv1.2     90 Application Data
    381 5.195814     192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=25 Win=501 Len=0 TSval=121973189 TSecr=3511419079
    499 7.203442     5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    500 7.203478     192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=82 Win=501 Len=0 TSval=121975197 TSecr=3511421087
    636 9.821692     5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    637 9.821736     192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=139 Win=501 Len=0 TSval=121977815 TSecr=3511423699
    685 10.717811    5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    686 10.717833    192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=196 Win=501 Len=0 TSval=121978711 TSecr=3511424601
    713 11.024359    5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    714 11.024382    192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=253 Win=501 Len=0 TSval=121979018 TSecr=3511424904
    750 11.620879    5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    751 11.620901    192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=310 Win=501 Len=0 TSval=121979614 TSecr=3511425504
    790 12.224671    5.196.95.208     192.168.33.133   TLSv1.2    123 Application Data
    791 12.224712    192.168.33.133   5.196.95.208     TCP         66 57932 → 8883 [ACK] Seq=25 Ack=367 Win=501 Len=0 TSval=121980218 TSecr=3511426107

> Frame 369: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
> Ethernet II, Src: Raspberr_79:fa:4a (dc:a6:32:79:fa:4a), Dst: CheckPoi_3a:01:49 (00:1c:7f:3a:01:49)
> Internet Protocol Version 4, Src: 192.168.33.133, Dst: 5.196.95.208
> Transmission Control Protocol, Src Port: 57932, Dst Port: 8883, Seq: 1, Ack: 1, Len: 24
v Transport Layer Security
   v TLSv1.2 Record Layer: Application Data Protocol: mqtt
        Content Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 19
        Encrypted Application Data: 4a417ab5bbc66f50c0d3937bd246f129714607
        [Application Data Protocol: mqtt]
```

9. Using a single Raspberry Pi 4 with the Online Broker

If only a single Raspberry is used, the adaptation is really simple.



Start the Runtime on your Raspberry then launch the straton Editor.

Follow the steps described in the previous parts but group all blocks in the same program:

- MQTTCONNECT, MQTTSUBSCRIBE, MQTTPUBSTRING and MQTTREADSTRING
- PRINTF blocks displaying the received message

Then, once the program is running, the Runtime displays the data sent/received on rising edge of the MQTTPUBSTRING.IN input.

10. Create and configure an application

10.1. Create a configuration

Open the IO Drivers window ( ), insert a configuration ( ) and select the "MQTT Client" configuration. The configuration is entered in a 4-level tree:

- Root: the MQTT protocol
  - A connection to a MQTT broker

Use the toolbar button to insert a connection.

Below are the properties you need to fill for a connection:

Connection ID	Identifies the connection in the straton project. Any call to a MQTT function block will require this string as an input argument.
Server	
Address	IP address or URL of the broker
IP port	Port number
Login	
Client ID	Client identification in the broker
User name	Optional login name
Password	Optional password
Options	
Keep alive timer	Optional keep alive period in seconds
Clean session	Open a clean session
Buffer size	Maximum size of MQTT telegrams
Last Will and Testament message (LWT)	
Topic name	Topic of the LWT message
Contents	Payload of the LWT message (text)
QoS	Quality of service for the LWT message
Other	
MQTT version	Version of the MQTT protocol (3.1 or 3.1.1)

11. MQTT function blocks

11.1. mqttConnect

Establish or release a MQTT connection

Inputs:

IN : BOOL	Triggers the connection (see notes)
CnxID : STRING	Identifier of the connection in the fieldbus configuration

Outputs:

OK : BOOL	TRUE if the connection is established
Error : BOOL	TRUE in case of a failed or lost connection

Notes:

- The connection is established on a rising edge of the IN input, and release on a falling edge of IN. Both outputs are reset to FALSE when the IN input becomes TRUE. The, either OK or Error output becomes TRUE.
- In case of a failed or lost connection, there is no automatic retry of connection. You will have to trigger again the IN input for a new trial.

11.2. mqttSubscribe

Subscribe to a message.

Inputs:

IN : BOOL	Triggers the subscribe request (see notes
CnxID : STRING	Identifier of the connection in the fieldbus configuration
Topic : STRING	Message topic – may contain wild chars
QOS : DINT	Wished quality of service

Outputs:

OK : BOOL	TRUE if the request is successfully sent
Error : BOOL	TRUE if the request failed

Notes:

The request is sent on a rising edge of the IN input, and release on a falling edge of IN. Both outputs are reset to FALSE when the IN input becomes TRUE. The, either OK or Error output becomes TRUE.

Important: you can send only one request (subscribe or unsubscribe or publish) at a time.

11.3. mqttUnsubscribe

Unsubscribe from a message.

Inputs:

IN : BOOL	Triggers the subscribe request (see notes
CnxID : STRING	Identifier of the connection in the fieldbus configuration
Topic : STRING	Message topic – may contain wild chars

Outputs:

OK : BOOL	TRUE if the request is successfully sent
Error : BOOL	TRUE if the request failed

Notes:

The request is sent on a rising edge of the IN input, and release on a falling edge of IN. Both outputs are reset to FALSE when the IN input becomes TRUE. The, either OK or Error output becomes TRUE.

Important: you can send only one request (subscribe or unsubscribe or publish) at a time.

11.4. mqttPubString

Publishes a message. Payload is passed as a STRING variable.

Inputs:

IN : BOOL	Triggers the subscribe request (see notes
CnxID : STRING	Identifier of the connection in the fieldbus configuration
Topic : STRING	Message topic – may contain wild chars
Payload : STRING	Payload of the message
QOS : DINT	Wished quality of service
Retain : BOOL	TRUE if the message must be retained in the broker

Outputs:

OK : BOOL	TRUE if the request is successfully sent
Error : BOOL	TRUE if the request failed

Notes:

The request is sent on a rising edge of the IN input, and release on a falling edge of IN. Both outputs are reset to FALSE when the IN input becomes TRUE. The, either OK or Error output becomes TRUE.

Important: you can send only one request (subscribe or unsubscribe or publish) at a time.

11.5. mqttPubTxb

Publishes a message. Payload is passed as a text buffer.

Inputs:

IN : BOOL	Triggers the subscribe request (see notes
CnxID : STRING	Identifier of the connection in the fieldbus configuration
Topic : STRING	Message topic – may contain wild chars
PayloadTxb : DINT	Handle of a valid text buffer containing the message payload
QOS : DINT	Wished quality of service
Retain : BOOL	TRUE if the message must be retained in the broker

Outputs:

OK : BOOL	TRUE if the request is successfully sent
Error : BOOL	TRUE if the request failed

Notes:

The request is sent on a rising edge of the IN input, and release on a falling edge of IN. Both outputs are reset to FALSE when the IN input becomes TRUE. The, either OK or Error output becomes TRUE.

Important: you can send only one request (subscribe or unsubscribe or publish) at a time.

11.6. mqttReadString

Pump received messages. Payload is passed as a STRING.

Inputs:

CnxID : STRING	Identifier of the connection in the fieldbus configuration

Outputs:

OK : BOOL	TRUE if a message was received
Topic : STRING	Topic name of the received message
Payload : STRING	Payload of the received message
QOS : DINT	Quality of service of the received message
Retain : BOOL	TRUE if the message was retained in the broker
Dup : BOOL	TRUE if the message was re-sent by the broker

Notes:

The system can receive at most one message per PLC cycle. So it is not worth to loop calling this block.

11.7. mqttReadTxb

Pump received messages. Payload is passed as a text buffer.

Inputs:

CnxID : STRING	Identifier of the connection in the fieldbus configuration
PayloadTxb : DINT	Handle of a valid text buffer to be filled with the payload of the received message

Outputs:

OK : BOOL	TRUE if a message was received
Topic : STRING	Topic name of the received message
QOS : DINT	Quality of service of the received message
Retain : BOOL	TRUE if the message was retained in the broker
Dup : BOOL	TRUE if the message was re-sent by the broker

Notes:
The system can receive at most one message per PLC cycle. So it is not worth to loop calling this block.

11.8. mqttStatus

Get status and statistics about a MQTT connection.

Inputs:

CnxID : STRING	Identifier of the connection in the fieldbus configuration
Reset : BOOL	If TRUE, statistic counters are reset to 0

Outputs:

CnxStat : DINT	Connection status: 0 = OK 9999 = invalid connection identifier 1000 = waiting for connection 1001 = TCP connection error 1002 = timeout during connection 1003 = invalid telegram received other = MQTT connection error code
NLinkOK	Number of correct MQTT telegrams received
NLinkErr	Number of incorrect MQTT telegrams received
NTimeout	Number of requests failed on timeout
NRecOK	Number of received message correctly read by the logic
NRecLost	Number of received message lost (unread by the logic)
NPubOK	Number of successful publish requests
NPubErr	Number of failed publish requests

12. Frequently asked questions

When trying to subscribe to a topic from the Mosquitto broker, it says “Error: Connection refused”

Check that the broker is launched using the “`top`” command or “`ps -A | grep mosq`”.

If it is not launched, you can start it in verbose mode (allows debug) using “`mosquitto -v`”, then the subscription must be made from another shell.

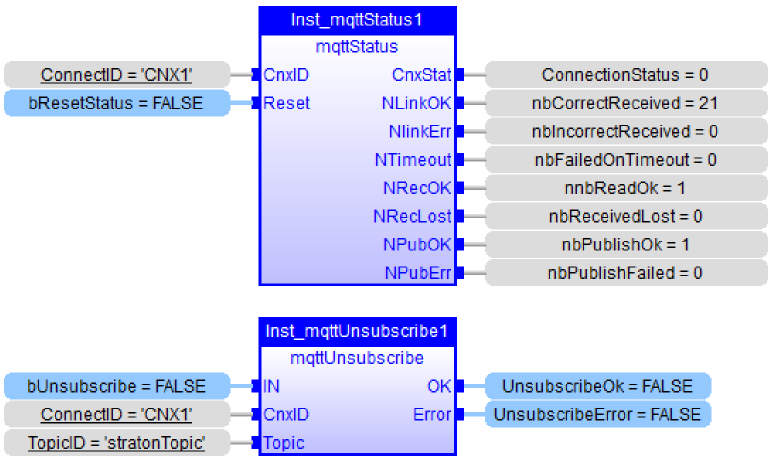
You can also start it in background using “`mosquitto -d`”.

To do a subscription to a topic use “`mosquitto_sub -d -t MyTopic`”.

If already launched, the application has an ID also called PID, displayed on the left of the top and ps commands. To kill the process, you have to kill the PID. For example: “`sudo kill 908`”

Are there other MQTT blocks available?

Yes, MQTTSTATUS block displays information about the connection and MQTTUNSUBSCRIBE allows to unsubscribe to a specific MQTT topic



How to check the encrypted communication between two raspberry with Tcpdump

You will install tcpdum on your Raspberry to save the frame. Do.
#sudo apt-get install tcpdump

#sudo tcpdump -w MyCapture104.pcapng

On this file the raspberry will write all the frame. Then you will need to take this file on your computer and open it with wireshark, you could then check that it’s encrypted.

You can download the file with FTP.