NetControl & MQTT over TLS (MQTTs)

Application Note

rev. 1.0

02.06.2022

Table of Contents

Document versions

Version	Date	Short description of changes
1.0	05.2014	Initial version

Symbols legend:

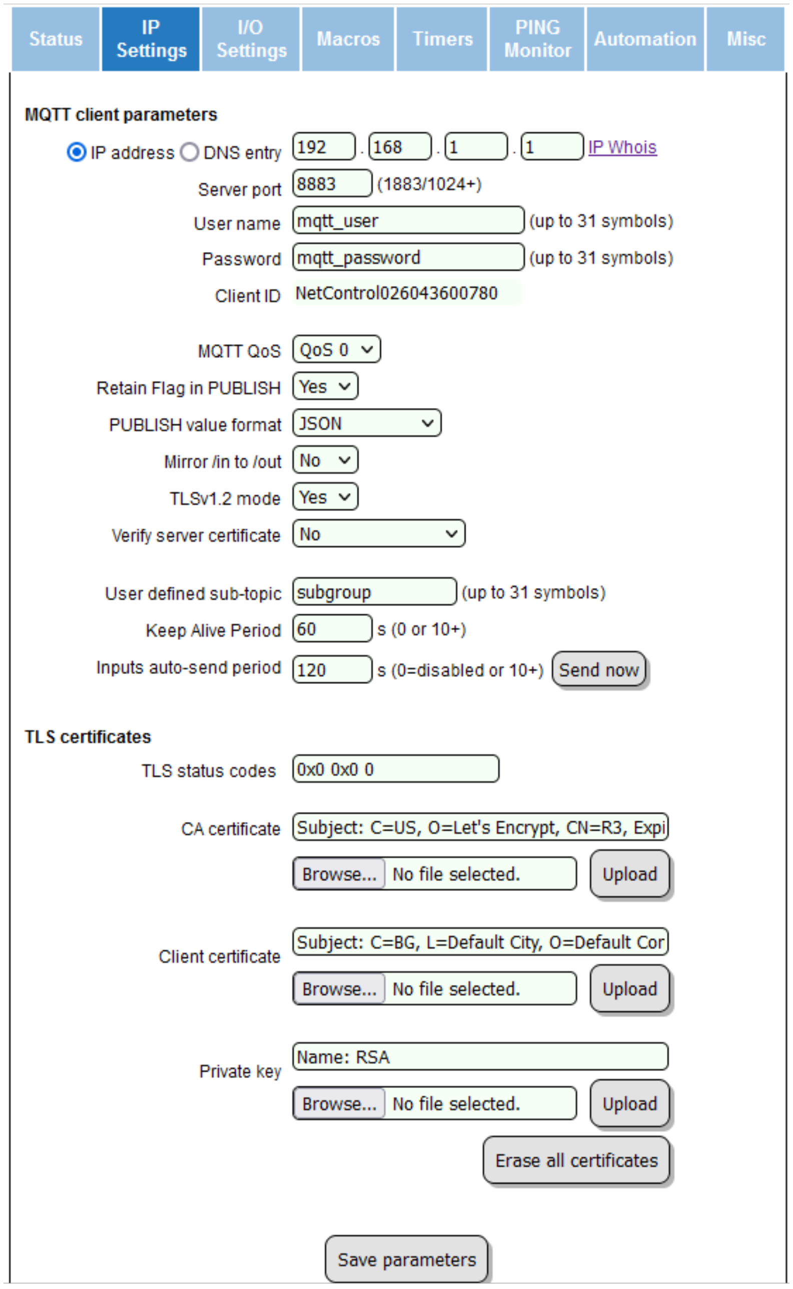
ⓘ *Text contains additional and useful information which explains more detailed some specific scenarios and specialties.*

⚡ *Text contains VERY IMPORTANT information that user must read!*

1. Introduction

NetControl supports the standard MQTT, which is not encrypted and is susceptible to eavesdropping and manipulation (especially on public networks). This problem is solved with MQTTS, which uses TLS as a transport layer.

In this version of the firmware we have added support for TLSv1.2 as a secure transport layer. It is implemented using the Mbed TLS.



MQTT client parameters

[✓ IP address] DNS entry	192 . 168 . 1 . 1 IP Whois
Server port	8883 (1883/1024+)
User name	mqtt_user (up to 31 symbols)
Password	mqtt_password (up to 31 symbols)
Client ID	NetControl026043600780
MQTT QoS	QoS 0
Retain Flag in PUBLISH	Yes
PUBLISH value format	JSON
Mirror /in to /out	No
TLSv1.2 mode	Yes
Verify server certificate	No
User defined sub-topic	subgroup (up to 31 symbols)
Keep Alive Period	60 s (0 or 10+)
Inputs auto-send period	120 s (0=disabled or 10+) Send now

TLS certificates

TLS status codes	0x0 0x0 0
CA certificate	Subject: C=US, O=Let's Encrypt, CN=R3, Expi
	Browse... No file selected. Upload
Client certificate	Subject: C=BG, L=Default City, O=Default Cor
	Browse... No file selected. Upload
Private key	Name: RSA
	Browse... No file selected. Upload

The following new parameters have been added to the MQTT settings menu:

- TLSv1.2 mode = Enable / Disable - you can select TLS or standard mode
- Verify Server Certificate - specify whether to validate the certificate provided by the server (you must have uploaded the appropriate CA certificate)
- CA certificate – Certification Authority certificate in PEM to certify the validity of the server certificate (if it will be verified)
- Client Certificate - a certificate in PEM format, which will authenticate the client to the server. This mode of operation is optional.
- Private Key - private key of the client certificate (it is obligatory if the authentication

mode with client certificate is used).

In case of validly read certificates, a part of their description will appear in the field next to them, ending with their expiration date. If there is a problem with the certificates, an error code from their decoding will be displayed (see Appendix I) .
For the private key, only "Name: RSA" is displayed with a valid key.

IMPORTANT!!! In order for the TLS to function correctly, you must have a correctly configured SNTP (Network Time) server with which ***NetControl*** *can synchronize its clock in real time. MQTTS is activated only after a correctly adjusted clock!*

NetControl *configuration file save / load function does NOT cover certificates!*

A firmware update deletes certificate information. Restoring factory IP settings also clears certificates.

Certificates requirements:

1. The server certificate must have a minimum of 2048 bit key
2. We recommend that the client certificate has a 1024 bit key. Longer lengths are also supported, but due to the lack of a hardware RSA accelerator, the time to verify the certificate in 2048 is over 5s (compared to 1.5s in 1024). This can affect the reliability of the TCP connection. Tests with 4096 bit key have not been performed!
3. Certificates and the private key must be in PEM text format (DER not supported). Each of the files must not be larger than 4096 bytes.

2. Supported security algorithms

TLS supports an extremely wide range of algorithms, but the implementation of all of them in devices with limited hardware and software resources is not justified. We have chosen a set in ***NetControl*** that has a relatively good level of security and is supported by a hardware accelerator.

It is possible to add other algorithms if necessary to solve specific tasks.

2.1. Hash functions

All SHA variants are supported: SHA1, SHA256, SHA384, SHA512. SHA1 and SHA256 are implemented with a hardware accelerator, so it is preferable to use SHA256 if possible (instead of SHA384 / 512).

The MD5 is also supported via a hardware accelerator, but is not recommended due to its low level of security.

2.2. Encryption algorithms

AES256 is supported via hardware accelerator (128, 196, 256).
DES / 3DES is not supported due to the low level of security.

2.3. Cipher Suites:

Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
Cipher Suite: TLS_RSA_WITH_AES_256_CCM (0xc09d)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_AES_256_CCM_8 (0xc0a1)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_128_CCM (0xc09c)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_AES_128_CCM_8 (0xc0a0)
Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)

2.4. Signature Hash Algorithms

Signature Algorithm: ecdsa_secp521r1_sha512 (0x0603)
Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
Signature Algorithm: ecdsa_secp384r1_sha384 (0x0503)
Signature Algorithm: rsa_pkcs1_sha384 (0x0501)
Signature Algorithm: ecdsa_secp256r1_sha256 (0x0403)
Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
Signature Algorithm: SHA224 ECDSA (0x0303)
Signature Algorithm: SHA224 RSA (0x0301)
Signature Algorithm: ecdsa_sha1 (0x0203)
Signature Algorithm: rsa_pkcs1_sha1 (0x0201)

2.5. Session Tickets

Session Tickets is a useful feature in TLS that is supported in the ***NetControl***. It speeds up the recovery of the connection after a short interruption (the full authorization procedure is not done through TLS, as information about the connection is exchanged between the client and the server, and re-used in the next establishment).

Times for initial establishment of the TLS connection (from „Client Hello“ to the first frame with MQTT data):

- without certificate verification: 275ms
- with certificate verification: 460ms
- with client certificate: 1400ms (1024 bit key), ~ 6000ms (2048bit)
- if available SESSION TICKET from previous connection: **55ms for all variants**

3. Generate self-signed certificates for Mosquitto

To generate the necessary server, CA and possibly client certificate, you can use the following manuals:

Mosquitto SSL Configuration -MQTT TLS Security:
http://www.steves-internet-guide.com/mosquitto-tls/

Creating and Using Client Certificates with MQTT and Mosquitto:
http://www.steves-internet-guide.com/creating-and-using-client-certificates-with-mqtt-and-mosquitto/

The most important things to pay attention to are:

1. In the case of a server certificate, the "Common Name" must match the IP address or DNS name of the server (depending on how you access it)
2. The private key of the client certificate should be WITHOUT a password
3. If you also use the client certificate to log in to the broker (use_identity_as_username = true), then the "Common Name" in it must be the name of the user who logs in.

4. Connecting to public servers

If you use the mode without checking the server's certificate, you should be able to freely connect to any public free / paid services.

There are many different testing options at https://test.mosquitto.org/.

In certificate verification mode, you will need to obtain the CA certificate through which to verify. Very often public services use a chain of CAs to check.

For example, one version of https://test.mosquitto.org/ uses a Lets Encrypt certificate.

Using the following command (needs OpenSSL package) you can get information about Certificate Chain:

```
>openssl s_client -connect test.mosquitto.org:8886
```

```
---
Certificate chain
 0 s:/CN=test.mosquitto.org
   i:/C=US/O=Let's Encrypt/CN=R3
 1 s:/C=US/O=Let's Encrypt/CN=R3
   i:/C=US/O=Internet Security Research Group/CN=ISRG Root X1
 2 s:/C=US/O=Internet Security Research Group/CN=ISRG Root X1
   i:/O=Digital Signature Trust Co./CN=DST Root CA X3
---
```

It can be seen that the server certificate is issued by the "R3" certification node. Therefore, in the list of Root and CA certificates of Lets Encrypt (https://letsencrypt.org/certificates/) you need to find the CA certificate of "R3" and load it in ***NetControl*** . Information can also be retrieved after decoding the server certificate itself.

5. Appendix I. Error codes

In the Status codes field you can see the last two errors of the TLS process and thus get a guide to the cause of a possible problem with a failed connection to the server. The same applies to decoding uploaded certificates (their error codes are in the table (*X509 Error codes*) and are displayed in the certificate information field).

```
/*
 * SSL Error codes
 */
/** The requested feature is not available. */
#define MBEDTLS_ERR_SSL_FEATURE_UNAVAILABLE               -0x7080
/** Bad input parameters to function. */
#define MBEDTLS_ERR_SSL_BAD_INPUT_DATA                    -0x7100
/** Verification of the message MAC failed. */
#define MBEDTLS_ERR_SSL_INVALID_MAC                       -0x7180
/** An invalid SSL record was received. */
#define MBEDTLS_ERR_SSL_INVALID_RECORD                    -0x7200
/** The connection indicated an EOF. */
#define MBEDTLS_ERR_SSL_CONN_EOF                          -0x7280
/** An unknown cipher was received. */
#define MBEDTLS_ERR_SSL_UNKNOWN_CIPHER                    -0x7300
/** The server has no ciphersuites in common with the client. */
#define MBEDTLS_ERR_SSL_NO_CIPHER_CHOSEN                  -0x7380
/** No RNG was provided to the SSL module. */
#define MBEDTLS_ERR_SSL_NO_RNG                            -0x7400
/** No client certification received from the client, but required by the authentication mode. */
#define MBEDTLS_ERR_SSL_NO_CLIENT_CERTIFICATE             -0x7480
/** Our own certificate(s) is/are too large to send in an SSL message. */
#define MBEDTLS_ERR_SSL_CERTIFICATE_TOO_LARGE             -0x7500
/** The own certificate is not set, but needed by the server. */
#define MBEDTLS_ERR_SSL_CERTIFICATE_REQUIRED              -0x7580
/** The own private key or pre-shared key is not set, but needed. */
#define MBEDTLS_ERR_SSL_PRIVATE_KEY_REQUIRED              -0x7600
/** No CA Chain is set, but required to operate. */
#define MBEDTLS_ERR_SSL_CA_CHAIN_REQUIRED                 -0x7680
/** An unexpected message was received from our peer. */
#define MBEDTLS_ERR_SSL_UNEXPECTED_MESSAGE                -0x7700
/** A fatal alert message was received from our peer. */
#define MBEDTLS_ERR_SSL_FATAL_ALERT_MESSAGE               -0x7780
/** Verification of our peer failed. */
#define MBEDTLS_ERR_SSL_PEER_VERIFY_FAILED                -0x7800
/** The peer notified us that the connection is going to be closed. */
#define MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY                 -0x7880
/** Processing of the ClientHello handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CLIENT_HELLO               -0x7900
/** Processing of the ServerHello handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_SERVER_HELLO               -0x7980
/** Processing of the Certificate handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CERTIFICATE                -0x7A00
/** Processing of the CertificateRequest handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CERTIFICATE_REQUEST        -0x7A80
/** Processing of the ServerKeyExchange handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_SERVER_KEY_EXCHANGE        -0x7B00
/** Processing of the ServerHelloDone handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_SERVER_HELLO_DONE          -0x7B80
/** Processing of the ClientKeyExchange handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CLIENT_KEY_EXCHANGE        -0x7C00
/** Processing of the ClientKeyExchange handshake message failed in DHM / ECDH Read Public. */
#define MBEDTLS_ERR_SSL_BAD_HS_CLIENT_KEY_EXCHANGE_RP     -0x7C80
/** Processing of the ClientKeyExchange handshake message failed in DHM / ECDH Calculate Secret. */
#define MBEDTLS_ERR_SSL_BAD_HS_CLIENT_KEY_EXCHANGE_CS     -0x7D00
/** Processing of the CertificateVerify handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CERTIFICATE_VERIFY         -0x7D80
/** Processing of the ChangeCipherSpec handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_CHANGE_CIPHER_SPEC         -0x7E00
/** Processing of the Finished handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_FINISHED                   -0x7E80
/** Memory allocation failed */
#define MBEDTLS_ERR_SSL_ALLOC_FAILED                      -0x7F00
/** Hardware acceleration function returned with error */
#define MBEDTLS_ERR_SSL_HW_ACCEL_FAILED                   -0x7F80
/** Hardware acceleration function skipped / left alone data */
#define MBEDTLS_ERR_SSL_HW_ACCEL_FALLTHROUGH              -0x6F80
```

```
/** Processing of the compression / decompression failed */
#define MBEDTLS_ERR_SSL_COMPRESSION_FAILED                -0x6F00
/** Handshake protocol not within min/max boundaries */
#define MBEDTLS_ERR_SSL_BAD_HS_PROTOCOL_VERSION           -0x6E80
/** Processing of the NewSessionTicket handshake message failed. */
#define MBEDTLS_ERR_SSL_BAD_HS_NEW_SESSION_TICKET         -0x6E00
/** Session ticket has expired. */
#define MBEDTLS_ERR_SSL_SESSION_TICKET_EXPIRED            -0x6D80
/** Public key type mismatch (eg, asked for RSA key exchange and presented EC key) */
#define MBEDTLS_ERR_SSL_PK_TYPE_MISMATCH                  -0x6D00
/** Unknown identity received (eg, PSK identity) */
#define MBEDTLS_ERR_SSL_UNKNOWN_IDENTITY                  -0x6C80
/** Internal error (eg, unexpected failure in lower-level module) */
#define MBEDTLS_ERR_SSL_INTERNAL_ERROR                    -0x6C00
/** A counter would wrap (eg, too many messages exchanged). */
#define MBEDTLS_ERR_SSL_COUNTER_WRAPPING                  -0x6B80
/** Unexpected message at ServerHello in renegotiation. */
#define MBEDTLS_ERR_SSL_WAITING_SERVER_HELLO_RENEGO       -0x6B00
/** DTLS client must retry for hello verification */
#define MBEDTLS_ERR_SSL_HELLO_VERIFY_REQUIRED             -0x6A80
/** A buffer is too small to receive or write a message */
#define MBEDTLS_ERR_SSL_BUFFER_TOO_SMALL                  -0x6A00
/** None of the common ciphersuites is usable (eg, no suitable certificate, see debug messages). */
#define MBEDTLS_ERR_SSL_NO_USABLE_CIPHERSUITE             -0x6980
/** No data of requested type currently available on underlying transport. */
#define MBEDTLS_ERR_SSL_WANT_READ                         -0x6900
/** Connection requires a write call. */
#define MBEDTLS_ERR_SSL_WANT_WRITE                        -0x6880
/** The operation timed out. */
#define MBEDTLS_ERR_SSL_TIMEOUT                           -0x6800
/** The client initiated a reconnect from the same port. */
#define MBEDTLS_ERR_SSL_CLIENT_RECONNECT                  -0x6780
/** Record header looks valid but is not expected. */
#define MBEDTLS_ERR_SSL_UNEXPECTED_RECORD                 -0x6700
/** The alert message received indicates a non-fatal error. */
#define MBEDTLS_ERR_SSL_NON_FATAL                         -0x6680
/** Couldn't set the hash for verifying CertificateVerify */
#define MBEDTLS_ERR_SSL_INVALID_VERIFY_HASH               -0x6600
/** Internal-only message signaling that further message-processing should be done */
#define MBEDTLS_ERR_SSL_CONTINUE_PROCESSING               -0x6580
/** The asynchronous operation is not completed yet. */
#define MBEDTLS_ERR_SSL_ASYNC_IN_PROGRESS                 -0x6500
/** Internal-only message signaling that a message arrived early. */
#define MBEDTLS_ERR_SSL_EARLY_MESSAGE                     -0x6480
/** An encrypted DTLS-frame with an unexpected CID was received. */
#define MBEDTLS_ERR_SSL_UNEXPECTED_CID                    -0x6000
/** An operation failed due to an unexpected version or configuration. */
#define MBEDTLS_ERR_SSL_VERSION_MISMATCH                  -0x5F00
/** A cryptographic operation is in progress. Try again later. */
#define MBEDTLS_ERR_SSL_CRYPTO_IN_PROGRESS                -0x7000
/** Invalid value in SSL config */
#define MBEDTLS_ERR_SSL_BAD_CONFIG                        -0x5E80


/**
 * X509 Error codes (Certificates)
 */
/** Unavailable feature, e.g. RSA hashing/encryption combination. */
#define MBEDTLS_ERR_X509_FEATURE_UNAVAILABLE              -0x2080
/** Requested OID is unknown. */
#define MBEDTLS_ERR_X509_UNKNOWN_OID                      -0x2100
/** The CRT/CRL/CSR format is invalid, e.g. different type expected. */
#define MBEDTLS_ERR_X509_INVALID_FORMAT                   -0x2180
/** The CRT/CRL/CSR version element is invalid. */
#define MBEDTLS_ERR_X509_INVALID_VERSION                  -0x2200
/** The serial tag or value is invalid. */
#define MBEDTLS_ERR_X509_INVALID_SERIAL                   -0x2280
/** The algorithm tag or value is invalid. */
#define MBEDTLS_ERR_X509_INVALID_ALG                      -0x2300
/** The name tag or value is invalid. */
#define MBEDTLS_ERR_X509_INVALID_NAME                     -0x2380
/** The date tag or value is invalid. */
#define MBEDTLS_ERR_X509_INVALID_DATE                     -0x2400
/** The signature tag or value invalid. */
#define MBEDTLS_ERR_X509_INVALID_SIGNATURE                -0x2480
/** The extension tag or value is invalid. */
#define MBEDTLS_ERR_X509_INVALID_EXTENSIONS               -0x2500
/** CRT/CRL/CSR has an unsupported version number. */
```

```
#define MBEDTLS_ERR_X509_UNKNOWN_VERSION                  -0x2580
/** Signature algorithm (oid) is unsupported. */
#define MBEDTLS_ERR_X509_UNKNOWN_SIG_ALG                  -0x2600
/** Signature algorithms do not match. (see \c ::mbedtls_x509_crt sig_oid) */
#define MBEDTLS_ERR_X509_SIG_MISMATCH                     -0x2680
/** Certificate verification failed, e.g. CRL, CA or signature check failed. */
#define MBEDTLS_ERR_X509_CERT_VERIFY_FAILED               -0x2700
/** Format not recognized as DER or PEM. */
#define MBEDTLS_ERR_X509_CERT_UNKNOWN_FORMAT              -0x2780
/** Input invalid. */
#define MBEDTLS_ERR_X509_BAD_INPUT_DATA                   -0x2800
/** Allocation of memory failed. */
#define MBEDTLS_ERR_X509_ALLOC_FAILED                     -0x2880
/** Read/write of file failed. */
#define MBEDTLS_ERR_X509_FILE_IO_ERROR                    -0x2900
/** Destination buffer is too small. */
#define MBEDTLS_ERR_X509_BUFFER_TOO_SMALL                 -0x2980
/** A fatal error occurred, eg the chain is too long or the vrfy callback failed. */
#define MBEDTLS_ERR_X509_FATAL_ERROR                      -0x3000
/* \} name */
```