Introduction

This application note describes IoT Cloud connectivity solution in general; introduces you briefly to IoT Cloud providers, like Amazon Web Services (AWS); and covers the Synergy MQTT/TLS module, its features, and operational flow sequence (Initialization/Data flow). The application example provided in the package uses AWS IoT Core. The detailed steps in this document show first-time AWS IoT Core users how to configure the AWS IoT Core platform to run this application example demonstration.

This application note enables you to effectively use the Synergy MQTT/TLS modules in your own design. Upon completion of this guide, you will be able to add the MQTT/TLS module to your own design, configure it correctly for the target application, and write code using the included application example code as a reference and efficient starting point. References to more detailed API descriptions, and other application projects that demonstrate more advanced uses of the module, are in the Synergy Software Package (SSP) User’s Manual (see Next Steps section), and serve as valuable resources in creating more complex designs.

Currently, the Synergy MQTT/TLS Connectivity solution is implemented and tested using AWS IoT Core on PK-S5D9/AE-Cloud2 kit. Support for other Synergy kits and IoT Cloud providers will be provided in upcoming releases.

Required Resources
To build and run the MQTT/TLS application example, you need:

Development tools & software
- Synergy Software Package (SSP) 1.5.0 or later (https://www.renesas.com/us/en/products/synergy/software/ssp.html)
- SEGGER J-Link® USB driver (renesassynergy.com/devtools/jlink)

Hardware
- Renesas Synergy™ Application Example kit (renesassynergy.com/kits/ae-wifi1)
- PC running Windows® 7 or 10; the Tera Term console, or similar application, and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari).
- Micro USB cables
- Ethernet cable

Prerequisites and Intended Audience
This application note assumes that you have some experience with the Renesas e² studio ISDE and Synergy Software Package (SSP). Before you perform the procedures in this application note, follow the procedure in the SSP User Manual to build and run the Blinky project. Doing so enables you to become familiar with the e² studio and the SSP, and validates that the debug connection to your board functions properly. In addition, this application note assumes you have some knowledge of MQTT/TLS and its communication protocols.

The intended audience are users who want to develop applications with MQTT/TLS modules using Renesas Synergy™ S5 or S7 MCU Series.
## Contents

1. Introduction to Cloud Connectivity .................................................................4  
   1.1 Overview ........................................................................................................ 4  
   1.2 Major Components ....................................................................................... 4  
   1.3 Cloud Provider Overview .............................................................................. 5  
      1.3.1 Amazon Web Services IoT Core ............................................................... 5  
   1.4 MQTT Protocol Overview ............................................................................ 6  
   1.5 TLS Protocol Overview ................................................................................ 7  
      1.5.1 Device Certificates, and Keys ................................................................. 7  
      1.5.2 Device Security Recommendations ....................................................... 8  
2. Synergy MQTT/TLS Cloud Solution ................................................................. 9  
   2.1 MQTT Client Overview .............................................................................. 9  
   2.2 Design Considerations ............................................................................... 9  
      2.2.1 Supported Features ............................................................................... 9  
      2.2.2 Operational Flow Sequence ................................................................. 9  
   2.3 TLS Session Overview .............................................................................. 10  
      2.3.1 Design Considerations .......................................................................... 10  
      2.3.2 Supported Features ............................................................................. 10  
      2.3.3 Operational Flow Sequence ................................................................. 11  
3. MQTT/TLS Application Example ................................................................. 13  
   3.1 Application Overview ............................................................................... 13  
   3.2 Software Architecture Overview ............................................................... 14  
      3.2.1 Console Thread ...................................................................................... 15  
      3.2.2 MQTT Thread ....................................................................................... 15  
      3.2.3 MQTT Rx Thread .................................................................................. 15  
   3.3 IoT Cloud Configuration (AWS) ................................................................. 15  
      3.3.1 Creating a Device on AWS IoT Core ................................................... 16  
      3.3.2 Generating Device Certificate and Keys .......................................... 19  
      3.3.3 Creating a Policy for your Device ..................................................... 20  
      3.3.4 Connecting the Certificate to the Policy ........................................... 21  
4. Running the MQTT/TLS Application .............................................................. 22  
   4.1 Importing, Building, and Loading the Project ............................................. 22  
   4.2 Manually Adding the Board Support Package for the AE-CLOUD2 Kit ....... 22  
   4.3 Powering up the Board ............................................................................. 23  
   4.4 Connect to AWS IoT Cloud ...................................................................... 23  
      4.4.1 Configuration Wizard Menu ................................................................. 24  
      4.4.2 Demo Start/Stop Command ................................................................. 34  
   4.5 Verifying the Demo .................................................................................... 35  
      4.5.1 Opening the MQTT Client on AWS IoT Core .................................... 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.2</td>
<td>Running the Synergy Cloud Connectivity Demonstration</td>
<td>37</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Monitoring MQTT Messages on AWS MQTT Client</td>
<td>37</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Publishing the MQTT Message from AWS MQTT Client</td>
<td>38</td>
</tr>
<tr>
<td>4.5.5</td>
<td>Stopping the Synergy Cloud Connectivity Demonstration</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>Next Steps</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>MQTT/TLS Reference</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Known Issues and Limitations</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>Appendix</td>
<td>39</td>
</tr>
<tr>
<td>8.1</td>
<td>Using your own Certificate with AWS IoT</td>
<td>39</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Registering your First CA certificate</td>
<td>39</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Registering a Device Certificate Signed by your CA Certificate</td>
<td>40</td>
</tr>
</tbody>
</table>

Revision History | 42 |
1. Introduction to Cloud Connectivity

1.1 Overview

Internet of Things (IoT) is a sprawling set of technologies described as connecting everyday objects, like sensors or smart-phones, to the World Wide Web. IoT devices are intelligently linked together to enable new forms of communication between things and people, and among things.

These devices, or things, connect to the network. Using sensors, they provide the information they gather from the environment or allow other systems to reach out and act on the world through actuators. In the process, IoT devices generate massive amounts of data, and cloud computing provides a pathway, enabling data to travel to its destination.

1.2 Major Components

The IoT Cloud Connectivity Solution includes the following major components:

1. Devices or Sensors
2. Gateway
3. IoT Cloud services
4. End user application/system

![IoT Cloud Connectivity Architecture](image)

**Devices or Sensors**

A device includes hardware and software that interacts directly with the world. Devices connect to a network to communicate with each other, or to centralized applications. Devices may connect to the Internet either directly or indirectly.

**Gateway**

A gateway enables devices that are not directly connected to the Internet to reach cloud services. The data from each device is sent to the Cloud Platform, where it is processed and combined with data from other devices, and potentially with other business-transactional data. Most of the common communication gateways support one or more communication technologies such as Wi-Fi, Ethernet, or Cellular.

**IoT Cloud**

Many IoT devices produce lots of data. You need an efficient, scalable, affordable way to manage those devices, handle all that information, and make it work for you. When it comes to storing, processing, and analyzing data, especially big data, it is hard to surpass the cloud.
1.3 Cloud Provider Overview

1.3.1 Amazon Web Services IoT Core

The AWS IoT Core service provides secure, bi-directional communication between IoT devices and the AWS Cloud over MQTT, HTTPS, and Web Sockets, enabling you to collect telemetry from multiple things, store the data, and analyze it.

AWS IoT Core is authenticated using TLS mutual authentication with X.509 certificates. Once a certificate is provisioned and activated, it can be installed on a device that then uses the certificate for all requests to device gateway. The following diagram summarizes the service components and the flow of data.

![AWS IoT Cloud Solution Diagram](image)

**Figure 2  AWS IoT Cloud Solution**

1.3.1.1 Key Features

1. **AWS IoT Device SDK**
   The AWS IoT Device SDK enables your devices to connect, authenticate, and exchange messages with the IoT Core using MQTT, HTTPS, or Web Sockets protocols. The AWS IoT Device SDK supports C, C++, Java, Node.JS, Python, and Arduino Yun. It includes the client libraries, the developer guide, and the porting guide for manufacturers.

2. **Device Gateway**
   The AWS IoT Device Gateway enables devices to securely and efficiently communicate with the AWS IoT Core. The Device Gateway can exchange messages using a publish-subscribe model, enabling one-to-one and one-to-many communications. With this one-to-many communication pattern, the AWS IoT Core makes it possible for a connected device to broadcast data to multiple subscribers for a given topic.
   The Device Gateway supports MQTT, Web Sockets, and HTTPS 1.1 protocols. The Device Gateway scales automatically to support over a billion devices without provisioning infrastructure.

3. **Authentication and Authorization**
   AWS IoT Core offers mutual authentication and encryption at all points of connection, so that data is never exchanged between devices and the AWS IoT Core without a proven identity. The AWS IoT Core supports the AWS method of authentication (called ‘SigV4’), X.509 certificate-based authentication, and customer created token based authentication (through custom authorizers). Connections using HTTPS can use any of these methods, while connections using MQTT use certificate-based authentication, and connections using Web Sockets can use SigV4 or custom authorizers.
(4) Registry

The Registry establishes an identity for devices and tracks metadata such as the devices’ attributes and capabilities. To each device, the Registry assigns a unique identity that is consistently formatted, regardless of the type of device or how it connects. The Registry also supports metadata that describes the capabilities of a device, for example, whether a sensor reports temperature data, and if the temperature scale is Fahrenheit or Celsius.

(5) Device Shadows

With the AWS IoT Core, you can create a persistent, virtual version, or a “shadow,” of each device that includes the device’s latest state. Applications, or other devices, can read these messages and interact with the device. Device Shadows persist in reporting the last state and the desired future state of each device, even when the device is offline. You can retrieve the device’s last reported state, or set a desired future state, through the API, or using the Rules Engine.

(6) Rules Engine

The Rules Engine makes it possible to build IoT applications that gather, process, analyze, and act on data generated by connected devices at a global scale, without having to manage any infrastructure. The Rules Engine evaluates inbound messages published into the AWS IoT Core and then transforms and delivers them to another device or a cloud service, based on business rules you define. A rule can apply to data from one or many devices, and it can take one or many actions in parallel.

The Rules Engine can also route messages to AWS endpoints, including AWS Lambda, Amazon Kinesis streams, Amazon S3, Amazon DynamoDB, Amazon Cloudwatch, and Amazon Elasticsearch with built-in Kibana integration. External endpoints can be reached using AWS Lambda, Amazon Kinesis, and Amazon Simple Notification Service (SNS).

1.4 MQTT Protocol Overview

MQTT stands for MQ Telemetry Transport. MQTT is a Client Server publish-subscribe messaging transport protocol. It is an extremely lightweight, open, simple messaging protocol, designed for constrained devices, as well as low-bandwidth, high-latency, or unreliable networks. These characteristics make it ideal for use in many situations, including constrained environments, such as communication in Machine to Machine (M2M) and IoT contexts, where a small code footprint is required, and/or network bandwidth is at a premium.

A MQTT client can publish information to other clients through a broker. A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for authentication, authorization of clients, as well as delivering published messages to any of its clients who subscribe to the topic. In this publisher/subscriber model, multiple clients may publish data with the same topic. A client will receive the messages published if the client subscribes to the same topic.

---

![Figure 3 MQTT Client Publish/Subscribe Model](image-url)
In this model, there is no direct connection between a publisher and the subscriber. To handle the challenges of a pub/sub system, the MQTT generally uses quality of service (QoS) levels. There are three QoS levels in MQTT:

- **At most once (0)**
- **At least once (1)**
- **Exactly once (2)**

**At most once (0)**
A message will not be acknowledged by the receiver or stored and redelivered by the sender.

**At least once (1)**
It is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once. The sender will store the message until it gets an acknowledgment in form of a PUBACK command message from the receiver.

**Exactly once (2)**
It is guaranteed that each message is received only once by the counterpart. It is the safest and the slowest quality of service level. The guarantee is provided by two flows there and back, between sender and receiver.

AWS IoT Core does not support QoS level 2.

### 1.5 TLS Protocol Overview

Transport Layer Security (TLS) protocol and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over a computer network.

The TLS/SSL protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

- **Encryption**: The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanisms such as AES (Advanced Encryption Standard) is used for data encryption.

- **Authentication**: A mechanism to check the peer’s identity using certificates.

- **Integrity**: A mechanism to detect message tampering and forgery to ensure that connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity.

TLS/SSL uses TCP but provides secure communication for application layer protocols, such as HTTP and MQTT.

![Figure 4 TLS/SSL Hierarchy](image)

### 1.5.1 Device Certificates, and Keys

Devices certificates, public and private keys, and the ways they can be generated, is discussed in this section.
1.5.1.1 Device Certificates

Security is a critical concern when deploying and managing IoT devices. In general, each of the IoT devices needs an identity before they can communicate with the cloud. Digital certificates are the most common method for authenticating a remote host in TLS. Essentially, a digital certificate is a document with specific formatting that provides identity information for a device.

TLS normally uses a format called X.509, a standard developed by the International Telecommunication Union, though other formats for certificates may be used, if the TLS hosts can agree on the format to be used. X.509 defines a specific format for certificates and various encoding that can be used to produce a digital document. Most X.509 certificates used with TLS are encoded using a variant of ASN.1, another telecommunication standard. Within ASN.1 there are various digital encodings, but the most common encoding for TLS certificates is the Distinguished Encoding Rules (DER) standard. DER is a simplified subset of the ASN.1 Basic Encoding Rules (BER) that is designed to be unambiguous, making parsing easier.

Though DER-formatted binary certificates are used in the actual TLS protocol, they may be generated and stored in a number of different encodings, with file extensions such as .pem, .crt, and .p12. The most common of the alternative certificate encodings is PEM. The PEM format (from Privacy-Enhanced Mail) is a base-64 encoded version of the DER encoding.

Depending on your application, you may generate your own certificates, be provided certificates by a manufacturer or government organization, or purchase certificates from a commercial certificate authority.

1.5.1.2 Loading Certificates onto your Device

To use a digital certificate in your NetX™ Secure application, you must first convert your certificate into a binary DER format, and optionally, convert the associated private key into a binary format; typically, a PKCS#1-formatted, DER-encoded RSA key. Once converted, it is up to you to load the certificate and the private key on to the device. Possible options include: using a flash-based file system or generating a C array from the data (using a tool such as “xd” from Linux® with the “-i” option) and compiling the certificate and key into your application as constant data.

Once your certificate is loaded on the device, you can use the TLS API to associate your certificate with a TLS session.

1.5.1.3 Generating Self-Signed Certificates

You may also choose to generate a self-signed certificate for testing purposes. The command to generate such a certificate is:

```bash
openssl req -x509 -newkey rsa:2048 -keyout private.key -out cert.pem -days 365 -nodes -subj "/C=US/ST=Oregon/L=Portland/O=Company Name/OU=Org/CN=www.example.com"
```

In this situation, a self-signed certificate “www.example.com” is generated. The certificate and private key files are cert.pem and private.key. You may generate a certificate for localhost by replacing www.example.com for localhost. In that case, pass localhost as the first-argument to the installation script.

1.5.2 Device Security Recommendations

The following security recommendations are not enforced by Cloud IoT Core but will help you secure your devices and connections.

- The private key should be kept secret.
- Use TLS 1.2 when communicating with IoT Cloud, and verify that the server certificate is valid using root certificate authorities.
- Each device should have a unique public/private key pair. If multiple devices share a single key and one of those devices is compromised, an attacker could impersonate all the devices that have been configured with that one key.
- Keep the public key secure when registering it with Cloud IoT Core. If an attacker can tamper with the public key and trick the provisioner into swapping the public key and registering the wrong public key, the attacker will subsequently be able to authenticate on behalf of the device.
- The key pair used to authenticate the device to Cloud IoT Core should not be used for other purposes or protocols.
- Depending on the device’s ability to store keys securely, key pairs should be rotated periodically. When practical, all keys should be discarded when the device is reset.
- If your device runs an operating system, make sure you have a way to securely update it. Android Things provides a service for secure updates. For devices that do not have an operating system, ensure that you can securely update the device’s software if security vulnerabilities are discovered after deployment.
2. Synergy MQTT/TLS Cloud Solution

2.1 MQTT Client Overview
The NetX Duo MQTT Client module provides high-level APIs for a Message Queuing Telemetry Transport (MQTT) protocol based client. The MQTT protocol works on top of TCP/IP and therefore the MQTT client is implemented on top of NetX Duo IP and NetX Duo Packet pool. NetX Duo IP attaches itself to the appropriate link layer frameworks, such as Ethernet, Wi-Fi, or cellular.

The NetX Duo MQTT client module can be used in normal or in secure mode. In normal mode, the communication between the MQTT client and broker is not secure. In secure mode, the communication between the MQTT Client and broker is secured using the TLS protocol.

2.2 Design Considerations
- By default, the MQTT client does not use TLS; communication is not secure between a MQTT client and broker.
- The Synergy MQTT client does not add the NetX Duo TLS session block. It only adds NetX Duo TLS common block. This block defines/controls the common properties of NetX secure.
- It is the responsibility of the user/application code to create the TLS session, configure the security parameters, and load the relevant certificates manually under the TLS setup callback provided by `nxd_mqtt_client_secure_connect()` API.

2.2.1 Supported Features
NetX Duo MQTT Client supports the following features:
- Provides an option to enable/disable TLS for secure communication using NetX Secure in SSP/.
- Supports QoS and provides the ability to choose the levels that can be selected while publishing the message.
- Internally buffers and maintains the queue of received messages.
- Provides a mechanism to register callback when a new message is received.
- Provides a mechanism to register callback when connection with the broker is terminated.

2.2.2 Operational Flow Sequence

```
Application             MQTT Client

Creates MQTT Client instance by calling `nxd_mqtt_client_create()`

Connect to MQTT broker using `nxd_mqtt_client_connect()`

Subscribe to topics using `nxd_mqtt_client_subscribe()` API

Publish to MQTT topics using `nxd_mqtt_client_publish()` API

Register callback on incoming MQTT messages using `nxd_mqtt_client_receive_notify_set()` API

`Receive_notify_callback()` invoked on incoming MQTT messages

Retrieve incoming MQTT messages using `nxd_mqtt_client_message_get()`

Disconnect MQTT client service using `nxd_mqtt_client_disconnect()`

Terminate MQTT client service using `nxd_mqtt_client_delete()`
```

Figure 5 Synergy MQTT Client Flow Sequence
2.3 TLS Session Overview

The NetX Duo TLS session module provides high-level APIs for the TLS protocol-based client. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo TLS Session module is based on Express Logic’s NetX Secure which implements the Secure Socket Layer (SSL) and its replacement, TLS protocol, as described in RFC 2246 (version 1.0) and 5246 (version 1.2). NetX Secure also includes routines for the basic X.509 (RFC 5280) format. NetX Secure is intended for applications using ThreadX RTOS in the project.

2.3.1 Design Considerations

- NetX Secure TLS performs only basic path validation on incoming server certificates.
  Once the basic path validation is complete, TLS then invokes the certificate verification callback supplied by the application.
- It is the responsibility of the application to perform any additional validation of the certificate.
  To help with the additional validation, NetX Secure provides X.509 routines for common validation operations, including DNS validation and Certificate Revocation List checking.
- Software-based cryptography is processor-intensive.
  NetX Secure software-based cryptographic routines have been optimized for performance, but depending on the power of the target processor, performance may result in very long operations. When hardware-based cryptography is available, it should be used for optimal performance of the NetX secure TLS.
- Due to the nature of embedded devices, some applications may not have the resources to support the maximum TLS record size of 16 KB.
  NetX Secure can handle 16 KB records on devices with sufficient resources.

2.3.2 Supported Features

- Support for RFC 2246 The TLS Protocol Version 1.0
- Support for RFC 5280 X.509 PKI Certificates (v3)
- Support for RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Transport Layer Security (TLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for TLS
2.3.3 Operational Flow Sequence
This section describes the TLS handshake operational sequence.

2.3.3.1 TLS Handshake
The following figure shows a typical TLS handshake between the TLS Server and Client.

![Figure 6 TLS Handshake](image)

- A TLS handshake begins when the TLS client sends a **ClientHello** message to a TLS server, indicating its desire to start a TLS session.
- The message contains information about the encryption the client would like to use for the session, along with information used to generate the session keys.
- The TLS server responds to the **ClientHello** with a **ServerHello** message, indicating a selection from the encryption options provided by the client.
- It is followed by a Certificate message, in which the server provides a digital certificate to authenticate its identity to the client.
- Finally, the server sends a **ServerHelloDone** message to indicate it has no more messages to send.
- Once Client has received all the server’s messages, it has enough information to generate the session keys. TLS does this by creating a shared bit of random data called the Pre-Master Secret, which is a fixed size and is used as a seed to generate all the keys needed once encryption is enabled.
• The Pre-Master Secret is encrypted using the public key algorithm (such as RSA) specified in the Hello messages and the public key provided by the server in its certificate.
• The encrypted Pre-Master Secret is sent to the server in the **ClientKeyExchange** message. The server, upon receiving the **ClientKeyExchange** message, decrypts the Pre-Master Secret using its private key and proceeds to generate the session keys in parallel with the TLS client.
• Once the session keys are generated, all further messages can be encrypted using the private-key algorithm (such as AES) selected in the Hello messages. One final un-encrypted message called **ChangeCipherSpec** is sent by both the client and server to indicate that all further messages will be encrypted.
• The first encrypted message sent by both the client and server is also the final TLS handshake message, called **Finished**. This message contains a hash of all the handshake messages received and sent. This hash is used to verify that none of the messages in the handshake have been tampered with or corrupted.
• Now the application begins sending and receiving data. All data — sent by either side — is first hashed using the hash algorithm chosen in the Hello messages, and then encrypted using the chosen private-key algorithm with the generated session keys.
• Finally, a TLS session can only be successfully ended if either the Client or Server chooses to do so. Both the client and server must send and process a **CloseNotify** alert for a successful session shutdown.

### 2.3.3.2 Initialization Flow Sequence

A typical TLS session initialization flow sequence is shown in the following figure.
2.3.3.3 Data Communication Flow Sequence

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a TCP connection using NetX/NetX Duo API</td>
</tr>
<tr>
<td>2</td>
<td>Establish connection using nx_tcp_server_socket_listen() and nx_tcp_server_socket_accept() services (for TLS Server mode) or the nx_tcp_client_socket_connect() service (for TLS Client mode)</td>
</tr>
<tr>
<td>3</td>
<td>Start the TLS session using nx_secure_tls_session_start() API</td>
</tr>
<tr>
<td>4</td>
<td>Allocate space for TLS header using nx_secure_tls_packet_allocate() API</td>
</tr>
<tr>
<td>5</td>
<td>Application send data using nx_secure_tls_session_send() API</td>
</tr>
<tr>
<td>6</td>
<td>Application receive data using nx_secure_tls_session_receive() API</td>
</tr>
<tr>
<td>7</td>
<td>Application close TLS session using nx_secure_tls_session_end() API</td>
</tr>
</tbody>
</table>

**Figure 8** Synergy TLS Session Data Flow Sequence

3. MQTT/TLS Application Example

3.1 Application Overview

This example application project demonstrates the Renesas Synergy™ IoT Cloud Connectivity solution using the onboard Synergy MQTT/TLS modules. For demonstration purposes, this application uses Amazon Web Services (AWS) as the cloud provider. Ethernet or Wi-Fi or Cellular (supported on AE-Cloud2 kit) is used as primary communication interface between the Thing and AWS IoT Core.

In this example, the PK-S5D9/AE-Cloud2 kit acts as MQTT node/Thing, connects to the AWS IoT Core, periodically reads its die temperature (in case of PK-S5D9 kit) or on-board sensors (in case of AE-Cloud2 kit) values and publishes it to the AWS IoT Core. It also subscribes to its User LED state MQTT topic. You can turn ON/OFF the User LEDs by publishing the LED state remotely. This application reads the updated LED state and turns ON/OFF the User LEDs.

The steps here use the MQTT Client from AWS IoT Core to subscribe to the MQTT topics published by PK-S5D9/AE-Cloud2 kit. Follow the instructions in section 3.3 to setup the MQTT client on AWS IoT Core and run this demonstration. However, you are free to use any known MQTT client to subscribe to the MQTT topics published by PK-S5D9/AE-Cloud2 Synergy MCU kit.
3.2 Software Architecture Overview

The following figure shows the software architecture for the Synergy Cloud Connectivity Application Example Project.

![Synergy Cloud Connectivity Application Software Architecture](image)

The main software components of this application are:

- MQTT Client
- NetX Duo IP Stack and its underlying driver components for Ethernet, Cellular and Wi-Fi.
- Console Framework

This application contains the following application threads:

- Console Thread
- MQTT Thread
- MQTT Rx Thread
3.2.1 Console Thread
This thread handles the function related to Common Line Interface (CLI). It uses the console framework, which in-turn, uses the communication framework and its underlying USBX CDC device module components.

This thread reads the user inputs and stores them in the internal data flash. The stored information is read later by the MQTT Thread when it tries to run the Synergy Cloud connectivity demo.

This thread presents you with the following CLI command options.

- `Cwiz`
- `Demo start/stop`

**Cwiz command option**
Using this command option, you can select the following configurations:
- Network interface such as Ethernet, Wi-Fi, and its associated IP mode (DHCP/Static).
- IoT cloud selection (AWS).
- Dump the existing configuration from flash.
- Exit the menu.

**Demo start/stop command option**
Using this command option, you can run/stop the Synergy Cloud Connectivity Demonstration.

3.2.2 MQTT Thread
This is the main control thread which thread handles the following major functions:
- Initialize communication interface (Ethernet/Wi-Fi).
- Initialize IoT Cloud interface.
- Read sensor data and publish the data periodically on MQTT topics.
- Update the user LED state based on the type of MQTT message received.

On wakeup, this thread periodically checks (every 5 seconds) for user input event flag state set once you enter the demonstration start/stop command on the CLI. If the demonstration start command is issued from the CLI, this thread will read the pre-configured user information from internal flash and checks its validity. If the content is valid, it then starts the Synergy Cloud connectivity demonstration. If a demo stop command is issued, it de-initializes the IoT cloud interface.

3.2.3 MQTT Rx Thread
This thread handles the incoming MQTT messages from the MQTT broker. On receiving the new MQTT message, the user callback `receive_notify_callback()` will be invoked by the MQTT thread. This callback in turn sets the semaphore on which the MQTT Rx Thread is polling periodically.

On receiving the new MQTT message, it uses the `nxd_mqtt_client_message_get()` API to read the message, parse it, and take action based on the type of the message received.

3.3 IoT Cloud Configuration (AWS)

**AWS IoT Policies**
AWS IoT Core policies are JSON (JavaScript Object Notation) documents that authorize your device to perform AWS IoT Core operations. AWS IoT defines a set of policy actions describing the operations and resources for which you can grant or deny access. For example:

- IoT:Connect represents permission to connect to the AWS IoT message broker.
- IoT:Subscribe represents permission to subscribe to an MQTT topic or topic filter.
- IoT:GetThingShadow represents permission to get a thing shadow.
**JSON**

JSON is an open standard, lightweight, data-interchange format. As a text document, it is easy for users to read and write, and machines to parse and generate.

JSON is completely language independent, but uses conventions that are familiar to C-family programmers, including C, C++, C#, Java, JavaScript, Perl, Python and many others.

```json
{
  "state": {
    "desired": {
      "LED_value": "On"
    }
  }
}
```

**AWS IoT Thing Shadow**

A Thing Shadow (also referred to as a Device Shadow) is a JSON document used to store and retrieve current state information for a Thing (device, application, and so on).

The Thing Shadow service maintains a thing shadow for each thing you connect to AWS IoT Core. You can use thing shadows to get and set the state of a thing over MQTT or HTTP, regardless of whether the thing is connected to the Internet. Each thing shadow is uniquely identified by its name.

**Amazon Web Services Signup**

Amazon Web Services offers a free account (12 months) for each user. It is expected that you created an account on the AWS IoT Cloud service before continuing to the next section.

To create an AWS account, open to the following link in your web browser:

https://portal.aws.amazon.com/billing/signup#/start

Fill in the required details and create a user account.

### 3.3.1 Creating a Device on AWS IoT Core

The following steps show you how to create a device on the AWS IoT Core user account. It is assumed that you created a user account in the AWS IoT Core and have followed the AWS signup procedure.

#### 3.3.1.1 Open AWS IoT Core Service

1. Connect to the AWS IoT service by typing AWS IoT in the AWS services search bar.
2. Click AWS IoT Core.
3.3.1.2 Create a Thing

1. Start creating your device by selecting Manage.
2. Then select Things.
3. Now select Create to create a thing.
4. Then select the Create a single thing button. It will open in a new window as shown in the following screen.
   Note: Remember to store the Thing Name. This information is passed to the firmware using serial console during configuration.

5. Enter the Thing Name, Thing Type, Attribute key, and value and press the Next button, as shown in the following screen. In the example, a Thing by name Thing_01 is created. Create a Thing type by clicking the Create a type button (such as Thing type: Synergy_PK); its attribute key is temperature and its attribute value is 25.
6. After you click Next button, it takes you to the next window shown in the following screen.
7. In this new window, click the option **Create thing without certificate** to create a thing in AWS IoT.
3.3.2 Generating Device Certificate and Keys

At this point, it is assumed the AWS IoT Thing has been created following the instructions in section 3.3.1. Now you can generate device certificates and keys for the AWS IoT Thing (Thing) created.

The Thing you created appear in Things section, as shown in the following screen.

1. Click the Thing you created. It will open in a new window with the Thing information.
   In the example, the Thing created is called Thing_01.
2. Go to the Security tab and click Create certificate button, as shown in the following screen.

   ![Security tab with Create certificate button](image)

   It generates the following for the Thing you created, as shown in the following screen.
   - A device certificate
   - A public key
   - A private key
   - A root CA for AWS IoT

3. To download certificates, click the **Download** button next to each of the certificates and keys, as shown in the following screen.
4. Click **Activate** button to activate the certificate just created.
5. After certificate activation, click the **Done** button to complete the certification/keys creation.

Note: The rootCA certificate for AWS is distributed as part of this package (rootCA.pem). Use this file for your testing.
3.3.3 Creating a Policy for your Device

At this point, it is assumed that the AWS IoT Thing has been created using the instructions in section 3.3.1 and device certificates have been generated by using the instructions in section 3.3.2.

1. Click the Secure option shown in the following screen.
2. Click the Policies option; it opens a window to create a new policy.
3. Click the Create button in the top right corner of the policies window to create new policy.

4. Enter the Name for your policy in the Name box as shown in the following screen.
5. Under Action, type: iot:*
6. Under Resource ARN, type: *
7. Click Allow
8. Click Create. Your policy has now been created.
3.3.4 Connecting the Certificate to the Policy

Before continuing, ensure that the Thing is created, certificates are generated, and the AWS IoT Policy is created using the instructions in sections 3.3.1 through 3.3.3.

1. Click the Secure option as shown in the following screen. Then click the Certificates option. It will open a window which lists the device certificates created in your AWS IoT Core service.
2. Choose the certificate you had created for your Thing. This can be done by clicking the “…” in the top right corner of your certificate.
3. Click Attach policy option from the drop-down menu.
4. Search for the policy from the Search policies window.
5. Choose the policy from the list and click the Attach button as shown in the following screen.
6. Your policy has now been attached to your device certificate.

4. Running the MQTT/TLS Application

4.1 Importing, Building, and Loading the Project
See the Renesas Synergy™ Project Import Guide (r11an0023eu0120-synergy-ssp-import-guide.pdf), included in this package, for instructions to import the project into e2 studio, build, and run the project.

4.2 Manually Adding the Board Support Package for the AE-CLOUD2 Kit
1. From the project bundle, locate the BSP file, Renesas.S5D9_PILLAR ARDUINO_MODULE.1.5.0.pack.
2. Copy the file (see Figure 10) to the following location:
   C:\Renesas\Synergy\e2studio_v6.2.1_ssp_v1.5.0\internal\projectgen\arm\packs.

![Figure 10 Load BSP Pack for AE-Cloud2 Kit](image)

Note: If e² studio is installed in any other location, the same information needs to be provided to copy the pack.
4.3 Powering up the Board

To connect power to the board, connect the SEGGER J-Link® debugger to the PC, connect the board to the PC USB port, and run the debug application, use the following instructions.

1. Connect the micro USB end of the supplied USB cable to the PK-S5D9 board J19 connector (DEBUG_USB) or AE-Cloud2 board J6 connector (DEBUG_USB).
   Note: The kit contains a SEGGER J-Link® On-board (OB). J-Link provides full debug and programming for the PK-S5D9/AE-Cloud2 board.
2. Connect the other end of the USB cable to the USB port on your workstation.
3. Attach the PMOD-based GT-202 Wi-Fi module in PMOD A connector.
4. In case of AE-Cloud2 kit, connect the BG96 Cellular shield on the AE-Cloud2 Arduino Connector, attach the Cellular antenna to the LTE antenna connector on the BG96 shield.

4.4 Connect to AWS IoT Cloud

The following instructions show how to run the Synergy Cloud connectivity application project and connect to the AWS IoT Cloud.

Note: At this stage, it is assumed you completed the instructions in section 3.3 to create an AWS IoT account, set up your device on the AWS IoT Core, and download the device certificates and keys.

1. If you have not already done so, complete section 3 and then go to 4.3 to power up and load the project on the PK-S5D9/AE-Cloud2 kit. Connect the USB Device port of the kit to the test PC and it will be automatically detected as an USB Serial device in case of Windows 10 PC. In case of Windows 7/8 PC, refer to the following installation guide to load the Synergy USB CDC driver. (https://www.renesas.com/en-us/products/synergy/software/add-ons/usb-cdc-drivers.html)
2. Open the serial console application, such as Tera Term, to connect it to the PK-S5D9/AE-Cloud2 kit. The default Tera Term settings are 8-N-1, and the baud rate is 9600.
3. Press Enter key in your keyboard; the following banner appears on the serial console.

---

**Figure 11** Welcome banner

4. Wait for the GPS initialization to complete. It may take around 7-10 seconds.
5. Press the ? key on your keyboard to display the available CLI command options as shown in the following figure.

---

**Figure 12** Help Menu
4.4.1 Configuration Wizard Menu

Enter command `cwiz` and press enter key in the serial console to enter the configuration menu. This command is used to configure the Network interfaces, AWS IoT Core Service, dump previous configuration stored in the internal flash.

**Figure 13  Configuration Menu**

4.4.1.1 Network Interface Selection

From the configuration menu, press ‘1’ key to configure the Network Interface. It lists the available network interface options in this application project. Currently this application supports Ethernet, Wi-Fi, Cellular (in case of AE-Cloud2 kit) communication interfaces.

**Figure 14  Network Interface Selection Menu**
(1) **Ethernet Network Interface Configuration**

From the **Network Interface Selection menu**, press ‘1’ key to select the Ethernet Network Configuration. You see the submenu where you choose the IP Address Configuration mode from the available options (DHCP/Static). Choose the **IP Address Configuration** mode. The selected Ethernet configuration setting is stored in internal flash; it is used at later stage, when communication is initialized.

![Ethernet Network Interface Configuration Menu](image1)

**Figure 15   Ethernet Network Interface Configuration Menu**

(2) **Wi-Fi Network Interface Configuration**

From the **Network Interface Selection menu**, press ‘2’ key to select the Wi-Fi Network Configuration.

![Wi-Fi Network Interface Configuration Menu](image2)

**Figure 16   Wi-Fi Network Interface Configuration Menu**
You are given the option to enter Wi-Fi Configuration settings, such as SSID, Passkey, Security type and IP Address Configuration mode.

The selected Wi-Fi configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.

![Wi-Fi Configuration](image)

Figure 17 Wi-Fi Configuration
(3) Cellular Network Interface Configuration

From the Network Interface Selection menu, press ‘3’ key to select the Cellular Network Configuration. You will be given two choices as shown in Figure 18.

Enter 1 in case of SIM provisioning. In this case, it is assumed that you already pre-configured the SIM card.
Enter 2 in case of SIM configuration. This option is ideal if you need to configure your brand-new SIM card using the AT shell interface.

![Figure 18 Cellular Configuration](image)

(a) Start Provisioning Option

In the cellular modem configuration menu, choose option 1 to enter the Start provisioning sub-menu as shown below.

![Figure 19 Cellular Modem Provisioning Menu](image)

You are given the option to enter Cellular Configuration settings, such as APN, Context ID, PDP type.
The selected Cellular configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.
(b) Start SIM Configuration Option

In the **Cellular modem configuration menu**, choose option 2 to enter the Start SIM Configuration sub-menu as shown below.

![Cellular Configuration Menu](image)

**Figure 20  Cellular Configuration Menu**

You can either choose option 1 to enter the Manual Configuration mode using AT command shell or choose option 2 to enter Auto Configuration from pre-stored AT command list. You generate this list at the end of previous Manual configuration.

Manual Configuration using AT Command Shell

In case you select option 1 in the Cellular Configuration Menu, enter the AT command shell as shown below. You can experiment various AT commands to configure the SIM cards.

![AT Command Shell](image)

**Figure 21  AT Command Shell**

Refer the following knowledge base article provided by Renesas as a baseline to provisioning the SIM card using the BG96 Cellular modem:

https://en.na4.teamsupport.com/knowledgeBase/18027787
To exit the AT command shell, enter the command “exit or EXIT”. You will be asked whether he wants to save the AT command as shown below.

If you chooses to save the AT commands, which can be later used to auto configure the new SIM cards, enter ‘Y’. In that case, you will be asked to enter the AT command details as shown below.

**Auto Configuration from pre-stored AT command list**

If you choose option 2 in the Cellular Configuration Menu, enter the Auto configuration from pre-stored AT command list menu as shown below.

> Figure 22  Auto configuration from pre-stored AT command list

The pre-stored AT commands will be sent to the cellular modem and their responses will be displayed in the console window.
4.4.1.2 AWS IoT Core Configuration

From the **Main Menu**, press ‘2’ and **Enter** key to configure the AWS IoT Core service as shown below.

![AWS IoT Core Configuration Menu](image)

**Figure 23**  AWS IoT Core Configuration Menu

1. **AWS IoT Core Setting Menu**

   From the AWS IoT Core configuration menu, press ‘1’ and **Enter** key to configure the AWS IoT Core settings as shown in the following graphic.

   ![AWS IoT Core Setting Menu](image)

   **Figure 24**  AWS IoT Core Setting Menu

   In the AWS IoT core configuration menu, you have the option of entering AWS Endpoint information and the AWS Thing name.
(2) AWS IoT Endpoint Information

To locate the AWS IoT endpoint information for your MQTT thing, use the following steps.

1. Open the thing you had created for this application. The thing created can be found under Manage tab, as shown in the following screen.

2. Open the Thing, go to the Interact tab as shown in the following screen. The AWS IoT Endpoint address can be found under the Rest API Endpoint block.

3. The selected configuration setting is stored in internal flash; it is used at later stage during the AWS IoT Core connection.

(3) AWS IoT Thing Name

To locate the name of your MQTT thing, use the following steps. In the following example, the MQTT thing name is Thing_01.
(4) Certificate/Keys Setting Menu

From the AWS IoT Core configuration menu, press ‘2’ and Enter key to configure the Device Certificate/Keys settings.

In the Device Certificate/Keys settings menu, you have the option of entering the root CA, device certificates, and device private key in .pem format.

Open these certificates in a text editor, copy and paste them in the serial console, and press Enter key as shown below.
The selected configuration setting is stored in the internal flash; it is used at later stage during the AWS IoT Core connection.

### 4.4.1.3 Dump Previous Configuration

From the **Main Menu**, choose option 3 to display the pre-selected network, the AWS IoT core Service Configuration options you selected from the internal flash, as shown in the following figure.
4.4.2 Demo Start/Stop Command
From the CLI console, enter `demo start` command to start the Synergy Cloud Connectivity Application Demonstration.

The application framework reads your pre-configured selection options for the network interface, the IoT Service from the internal flash, and checks for its validity. If the content is valid, it then initializes the network interface and establishes a MQTT connection with the AWS IoT Core.

This application wakes up periodically (every 5 seconds) and checks for your input event flag state. The flag state is set once you have entered the `demo start/stop` command on the CLI. This application does the following functions periodically until you enter the `demo stop` command.
1. Initialize communication interface (Ethernet/Wi-Fi/Cellular).
2. Initialize IoT Cloud interface.
3. Read sensor data and publish them periodically on MQTT topics.
4. Updates your LED state based on the type of MQTT message received.

If the demo stop command is issued, it de-initializes the IoT Cloud interface, which in turn stops MQTT messages from publishing and clears any pending MQTT messages from its internal queue.

4.5 Verifying the Demo

Use the following instructions to verify the functionality of this Synergy Cloud Connectivity Application Project.

Note: It is assumed you completed the instructions in section 3.3 to create an AWS IoT account, setup your device on AWS IoT Core, created the device certificates and keys, and compiled and downloaded the application project to the PK-S5D9/AE-Cloud2 kit.

Follow the instructions in section 4.4 to configure the network interface, IoT service configuration.

4.5.1 Opening the MQTT Client on AWS IoT Core

1. Login to your AWS IoT Core account and select your Thing that you created in section 3.3.

2. Click the Activity tab as shown in the following screen.

3. Click MQTT Client on the top right corner of the window highlighted in following figure.
The MQTT Client window opens on the AWS IoT Core. You can subscribe to the topics you want to listen and publish your MQTT message on the topic you want to publish.

4. Enter the Subscription topic in the Subscription topic box as shown in the following screen. Then click the Subscribe to topic button. The AWS MQTT Client starts listening to the topic to which you subscribed.

For this application, the following topic subscriptions were made:

**Get User LED state:**
$aws/things/<your thing name>/shadow/update

**Get Sensor data (telemetry data):**
“renesas/ae_cloud2/<your thing name>”
5. Also you can enter the MQTT topic they want to publish their message in the **Publish box** as shown in the following screen.

   *User LED state is published in the following topic:*
   
   \"$aws/things/<your thing name>/shadow/update/accepted\"

6. Enter the MQTT message in JSON format in the message window below the Publish button.

7. Next click the **Publish to topic** button to publish your MQTT message. Any MQTT clients who are subscribed to your MQTT publish topics will get your published message.

### 4.5.2 Running the Synergy Cloud Connectivity Demonstration

Run this application demonstration using the **demo start** command from the serial console.

Once you run **demo start**, it begins to configure the network interface, establish a connection with AWS IoT Core, and start publishing sensor data periodically (every 5 seconds).

### 4.5.3 Monitoring MQTT Messages on AWS MQTT Client

**Note:** At this stage, it is assumed that you followed **step 1** in section 4.5 and opened the MQTT Client window on the AWS IoT Cloud.

Once the demo is running on the kit, sensor data is periodically published to the AWS IoT Core. You must subscribe to the same MQTT Topics to receive this sensor data.

1. In this application, subscribe to the following topic to get the User LED state:
   
   \$aws/things/<your thing name>/shadow/update

2. Subscribe to the following topic to get sensor data:
   
   renesas/ae_cloud2/<your thing name>

3. Publish your LED state in the following topic:
   
   \$aws/things/<your thing name>/shadow/update/accepted

   Once you subscribed to the above MQTT topic, you should be able to see the sensor data published by the PK-SSD9/AE-Cloud2 kit, as shown in the following figure.
4.5.4 Publishing the MQTT Message from AWS MQTT Client

You can publish the MQTT message to turn ON/OFF your LEDs on the PK-S5D9/AE-Cloud2 kit by using the following table indicating messages for various LED states.

<table>
<thead>
<tr>
<th>LED State</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED LED ON</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Red_LED&quot;:&quot;ON&quot;}}}</td>
</tr>
<tr>
<td>RED LED OFF</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Red_LED&quot;:&quot;OFF&quot;}}}</td>
</tr>
<tr>
<td>YELLOW LED ON</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Yellow_LED&quot;:&quot;ON&quot;}}}</td>
</tr>
<tr>
<td>YELLOW LED OFF</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Yellow_LED&quot;:&quot;OFF&quot;}}}</td>
</tr>
<tr>
<td>GREEN LED ON</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Green_LED&quot;:&quot;ON&quot;}}}</td>
</tr>
<tr>
<td>GREEN LED OFF</td>
<td>`{&quot;state&quot;:{&quot;desired&quot;:{&quot;Green_LED&quot;:&quot;OFF&quot;}}}</td>
</tr>
</tbody>
</table>

4.5.5 Stopping the Synergy Cloud Connectivity Demonstration

To stop the demonstration, enter the `demo stop` command. Issuing this command de-initializes the IoT Cloud interface, stops it from publishing MQTT messages, and clears any pending MQTT messages from its internal queue.

---

5. Next Steps

Visit renesassynergy.com/tools to learn more about development tools & utilities.


Renesas Synergy Module Guides collateral link: https://www.renesas.com/en-us/products/synergy/tools-kits.htm#sampleCodes

6. MQTT/TLS Reference

- SSP 1.5.0 User’s Manual can be downloaded from the Renesas Synergy™ Gallery (http://www.renesassynergy.com/gallery)
- AWS IoT documentation (https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html)

7. Known Issues and Limitations

Note: Support for IAR EW with SSC will be available soon. For now, it is supported only on e2 studio.
If the user running this demo using Ethernet connection in some corporate network tries to stop the demo using `demo stop` command and then restart the demo using `demo start` command, the demo fails to reconnect to the AWS MQTT broker.

8. Appendix

8.1 Using your own Certificate with AWS IoT

These instructions walk you through an end-to-end process of setting up a client that is using a device certificate signed by your own CA.

Note: It is assumed that you are familiar with AWS IoT, and the process of creating an AWS IoT certificate. If you need more information about using AWS IoT generated certificates or want to learn more about authentication in AWS IoT, see the AWS IoT developer references in section 6.

1. Generate a CA certificate that will be used to sign your device certificate.
2. Next, register the CA certificate and register the device certificates.

Your device certificate now will be ready to connect the AWS IoT service.

8.1.1 Registering your First CA certificate

If you are a manufacturer, you would have purchased CA certificates from vendors, such as Symantec, Verisign, and so on. To use your own X.509 certificates that have been signed by your CA certificate, AWS IoT needs to verify not only that you own the CA certificate, but that you also have access to the private key for that certificate.

Start by creating your first sample CA certificate using `openssl` in a terminal. Actually, you would have the signing certificates issued by your CA vendor in the place of this sample CA. This sample CA certificate is used later in the walkthrough to sign a device certificate that you register with AWS IoT.

```
$ openssl genrsa -out deviceCertOne.key 2048
$ openssl req -new -key deviceCertOne.key -out deviceCertOne.csr
$ openssl x509 -req -in deviceCertOne.csr -CA sampleCACertificateOne.pem -CAkey sampleCACertificateOne.key -CAcreateserial -out deviceCertOne.crt -days 365 -sha256
```

Now that you have created a sample CA certificate, you need to register it with AWS IoT. When registering a CA certificate with AWS IoT, you follow a workflow to verify that you have access to both the CA certificate and the private key associated with the CA certificate. To verify ownership of the private key, you generate a verification certificate using the CA certificate, the private key, and a registration code that you generate from AWS IoT.

The registration workflow first requires retrieving a registration code from AWS IoT. Use the AWS CLI to generate a registration code with the following command:

```
$ aws iot get-registration-code
```

The AWS CLI command, `get-registration-code`, returns a randomly-generated unique registration code that is bound to your AWS account. This registration code is long-lived and does not expire until you delete it. To illustrate using the registration code, create a new CSR:

```
$ openssl genrsa -out privateKeyVerificationOne.key 2048
$ openssl req -new -in privateKeyVerificationOne.key -out privateKeyVerificationOne.csr
```

During the CSR process, you are prompted for information. Enter the registration code from AWS IoT into the Common Name field of the verification certificate.

The registration code validates that the generated verification certificate was created specifically for registering the CA certificate with AWS IoT, and that the verification certificate is not a previously issued certificate.

Now that you have a CSR that includes the registration code from AWS IoT, use your first sample CA certificate and the CSR to create a new certificate:

```
$ openssl x509 -req -in privateKeyVerificationOne.csr -CA sampleCACertificateOne.pem -CAkey sampleCACertificateOne.key -CAcreateserial -out privateKeyVerificationOne.crt -days 365 -sha256
```

When you register your CA certificate to AWS IoT, the combination of the registration code, the verification certificate signed with the CA private key, and the CA certificate are used to verify ownership of the CA private key.
Log into the AWS IoT console: Select **Use my certificate**, select **Register your CA certificate**, and then upload your sample CA certificate and verification certificate.

Once your CA certificate has been uploaded to AWS IoT, select the CA certificate from the console and select the **Actions** options to activate the CA certificate.

### 8.1.2 Registering a Device Certificate Signed by your CA Certificate

Now that you have created, registered, and activated a sample CA certificate; use the CA certificate to create a new device certificate and upload the device certificate into AWS IoT.

Enter the following commands in your terminal to create a device certificate:

```bash
$ openssl genrsa -out deviceCertOne.key 2048  
$ openssl req -new -key deviceCertOne.key -out deviceCertOne.csr  
$ openssl x509 -req -in deviceCertOne.csr -CA sampleCACertificateOne.pem -CAkey sampleCAACertificateOne.key -CAcreateserial -out deviceCertOne.crt -days 365 -sha256
```

You can upload the device certificate through the AWS Console.

In the AWS Console, navigate to AWS IoT and select **Use my certificate**, then **Upload existing device certificates**, to upload your device certificate into AWS IoT.

After uploading the device certificate, activate the device certificate in the AWS console so you can begin using the device certificate to communicate over MQTT with AWS IoT.

Once a device certificate is registered with AWS IoT, you can interact with it as in the case of a device certificate generated by AWS IoT. You can attach policies and associate things to your device certificate. You can also manage the lifecycle of that device certificate in AWS IoT, such as deactivate, revoke, or activate it in your AWS account.
Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software
- Software Package: renesassynergy.com/ssp
- Software add-ons: renesassynergy.com/addons
- Software glossary: renesassynergy.com/softwareglossary
- Development tools: renesassynergy.com/tools

Synergy Hardware
- Microcontrollers: renesassynergy.com/mcus
- MCU glossary: renesassynergy.com/mcuvglossary
- Parametric search: renesassynergy.com/parametric
- Kits: renesassynergy.com/kits

Synergy Solutions Gallery
- Partner projects: renesassynergy.com/partnerprojects
- Application projects: renesassynergy.com/applicationprojects

Self-service support resources:
- Documentation: renesassynergy.com/docs
- Knowledgebase: renesassynergy.com/knowledgebase
- Forums: renesassynergy.com/forum
- Training: renesassynergy.com/training
- Videos: renesassynergy.com/videos
- Chat and web ticket: renesassynergy.com/support
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep 24, 2018</td>
<td>-</td>
<td></td>
<td>Initial version</td>
</tr>
</tbody>
</table>

All trademarks and registered trademarks are the property of their respective owners.
Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the correctness or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers, office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machines tools; personal electronic equipment; industrial robots, etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; underwater repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

8. Contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be altered, modified, copied, or reverse engineered in any form, in whole or in part, without prior written consent of Renesas Electronics. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics. You are responsible for any losses or damages incurred by you or third parties arising from the use of these circuits, software, or information.

12. Refer to "http://www.renesas.com/" for the latest and detailed information.

(Rev 4.0-1 November 2017)