

## RX Family

R01AN6948EJ0200

Rev.2.00

Jan.23.2024

## TLS Implementation Example Using TSIP Driver (Azure RTOS)

### Important Notice:

On November 21, 2023, Microsoft announced that they have decided to contribute Azure RTOS to Open Source

under the stewardship of the Eclipse foundation and Azure RTOS becomes Eclipse ThreadX.

For detailed information, please refer to the announcement titled at Microsoft Contributes Azure RTOS to Open Source.

The support strategy scheme for Eclipse ThreadX will be determined and communicated at a later date.

Microsoft will discontinue the Azure RTOS and Azure RTOS Middleware under the existing agreement LICENSED-HARDWARE.txt.

It's important to note that updates for Azure RTOS on these hardware will no longer be provided.

### Introduction

The Trusted Secure IP (TSIP) driver supports APIs for SSL/TLS (referred to below as TLS) communication. This document presents an example of adding the TSIP driver to Azure RTOS, which includes NetX Duo, and explains how to confirm its operation. A sample project based on Azure RTOS is bundled with this document. The sample project adds the TSIP driver to Azure RTOS, which includes NetX Duo, and enables testing of MQTT communication via Microsoft Azure.

The sample project described in this document establishes connections via Microsoft Azure IoT Hub and IoT Hub Device Provisioning Service (DPS). TLS is used to authenticate connections with DPS.

### Devices on Which Operation Confirmed

The operation of the sample program appended to this document has been confirmed on the following devices.

RX65N: R5F565NEHDF

RX72N: R5F572NNHDFB

Note: The descriptions in this document use the CK-RX65N (Ethernet) as an example.

## Operating Environment

The operation of the sample program bundled with this document has been confirmed on the following environment.

IDE	<a href="#">e<sup>2</sup> studio 2023-10</a>
Toolchain	<a href="#">CCRX compiler v3.05.00</a>
Target board	CK-RX65N (Ethernet) CK-RX65N+RYZ014A (Cellular) Renesas Starter Kit+ for RX65N-2MB RX72N Envision Kit
Debugger	E2 Lite emulator (incorporated into the CK-RX65N board and RX72N Envision Kit)
Azure RTOS	v6.2.1_rel-rx-1.3.0
Driver package (RDP)	<a href="#">RX Driver Package V1.41</a>
TSIP driver	<a href="#">Version 1.19 (binary version)</a>
Security Key Management Tool	<a href="#">Security Key Management Tool V.1.04</a>
Tera Term	Version 4.106
OpenSSL	3.1.3 Light
Gpg4win (Kleopatra)	4.1.0
Shell script (bash) execution environment	Cygwin version 3.4.6

Note: This document provides a sample project for each supported target board.

Import the sample project that is appropriate for the target board you use.

Note that the CK-RX65N board comes in two versions: the Ethernet version, which uses an on-board LAN, and the Cellular version, which uses a mobile network that becomes available by connecting the RYZ014A cellular module to the board.

Also note that the applicable sample project differs depending on the version.

## Related Application Notes

For details of TLS communication using the TSIP, refer to the following application note.

- RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (Binary version) ([R20AN0548](#))

For a TLS implementation example using the TSIP driver and FreeRTOS, refer to the following application note.

- RX Family Implementing TLS Using TSIP Driver ([R01AN5880](#))

For a description of Azure operations, refer to the following document.

- RX65N Group Visualization of Sensor Data using RX65N Cloud Kit and Azure RTOS ([R01AN6011](#))

## Contents

1. Overview .....	5
1.1 Advantages of TLS Communication Using TSIP .....	5
1.2 TLS flow with TSIP .....	5
1.3 Cipher Suites Supported by TSIP Driver.....	5
1.4 TLS APIs of TSIP Driver.....	6
1.5 Definitions of Terms.....	7
2. Preparing the Sample Project.....	8
2.1 Creating a Workspace.....	9
2.2 Downloading the Project .....	10
2.3 Importing the Project .....	10
2.4 Key and Certificate Preparation .....	12
2.4.1 Installing OpenSSL.....	12
2.4.2 Obtaining Root CA Certificate .....	13
2.4.3 Generating RSA Keys and Client Certificate.....	20
2.4.4 Root CA Certificate Signature Generation and Certificate File Format Conversion .....	22
2.4.5 Key Wrapping and Registration in the Project .....	24
2.4.5.1 Creating a UFPK and W-UFPK.....	25
2.4.5.2 Wrapping the Keying Data .....	34
3. Operations on Microsoft Azure Portal .....	42
3.1 Preparations for Connection to Azure IoT Hub (Azure Portal).....	42
3.1.1 Creating an IoT Hub .....	42
3.1.2 Creating an IoT Hub Device Provisioning Service (DPS) Instance.....	42
3.1.3 Device Provisioning Using the IoT Hub and DPS Instance .....	42
3.2 Microsoft Azure Communication Settings .....	48
3.2.1 Azure IoT Settings.....	48
3.2.2 IP Address Settings.....	51
3.2.3 Client Certificate Format Selection.....	52
4. Building and Running the Project.....	53
4.1 Items to Confirm Before Building the Project .....	53
4.1.1 Settings for Renesas Starter Kit+ for RX65N-2MB .....	53
4.1.2 Setting for Code Generation During Build.....	53
4.2 Building the Project.....	55
4.3 Confirming Connection to Microsoft Azure.....	56
4.3.1 Checking Registration Status.....	56
4.3.2 Checking the Device .....	57
5. Appendix .....	58
5.1 Details of the Security Key Management Tool.....	58

5.2 TLS Communication Performance Using TSIP Driver ..... 58

6. Revision History.....59

Notes:

- Git® is a trademark of Software Freedom Conservancy, Inc. (<https://www.git-scm.com/about/trademark>)
- GitHub® is a trademark of GitHub, Inc. (<https://github.com/logos>)
- OpenSSL™ is a trademark of OpenSSL Software Foundation. (<https://www.openssl.org/policies/trademark.html>)
- Microsoft Azure is a trademark of Microsoft Corporation. (<https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks>)

## 1. Overview

### 1.1 Advantages of TLS Communication Using TSIP

The TSIP driver supports APIs for TLS. These APIs provide the following two advantages.

- No keying information is handled as plaintext during TLS protocol processing, thereby reducing the risk that customer keying information stored on the device may leak.
- Hardware acceleration speeds up encryption processing.

### 1.2 TLS flow with TSIP

The TLS flow in this sample project is shown below.

This flow is an example of using an X.509 self-signed certificate (when the key exchange method is RSA) for device authentication.

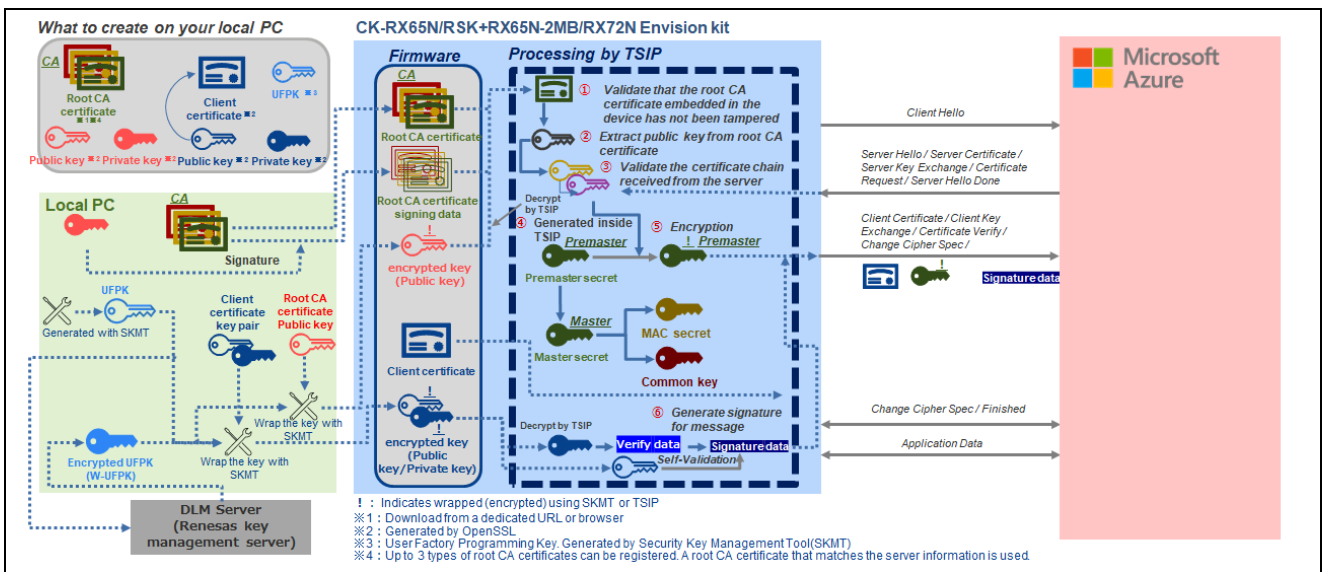


Figure 1-1 TLS flow with TSIP

### 1.3 Cipher Suites Supported by TSIP Driver

The TSIP driver supports the following cipher suites conforming to TLS 1.2.

- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256\*1
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256\*1
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256\*1

Note: 1. Server and client authentication (ECDSA) and encryption processing (AES-GCM) are not supported by the sample project described in this document.

### 1.4 TLS APIs of TSIP Driver

Table 1.1 lists the TSIP driver APIs used for TLS communication. For details of each API, the application note RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (Binary Version) ([R20AN0548](#)).

**Table 1-1 API Functions Used for TLS Communication**

Where Used	API Function
Certificate installation	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_Close R_TSIP_Open R_TSIP_TlsRootCertificateVerification
Random number generation	R_TSIP_GenerateRandomNumber
Client authentication (key data processing)	R_TSIP_GenerateRsa2048PrivateKeyIndex R_TSIP_GenerateRsa2048PublicKeyIndex
Handshake message hash calculation	R_TSIP_Sha256Init/Update/Final
Certificate	R_TSIP_TlsCertificateVerificationExtension R_TSIP_TlsCertificateVerification
Server key exchange Client key exchange (ECDHE key exchange algorithm)	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves R_TSIP_GenerateTlsP256EccKeyIndex R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key
Client key exchange (RSA key exchange algorithm)	R_TSIP_TlsGeneratePreMasterSecret R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey
Certificate Verify	R_TSIP_RsassaPkcs1024/2048SignatureGenerate R_TSIP_RsassaPkcs1024/2048SignatureVerification R_TSIP_EcdsaP192/224/256/384SignatureGenerate R_TSIP_EcdsaP192/224/256/384SignatureVerification
Finished	R_TSIP_TlsGenerateMasterSecret R_TSIP_TlsGenerateVerifyData R_TSIP_TlsGenerateSessionKey R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final
Application Data	R_TSIP_TlsGenerateSessionKey R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final

## 1.5 Definitions of Terms

Terms used in this document are defined below.

**Table 1-2 Terms**

<b>Terms</b>	<b>Description</b>
Key injection	Injecting a wrapped key into the device at the factory.
User key	An encryption key in plaintext used by the user. Not used on the device. For RSA and ECC, user keys are used as public keys and secret keys.
Encrypted key	Key information generated by encrypting a user key using a UFPK or update key and adding a MAC value. An encrypted key corresponding to the same user key is a shared value on each device.
Wrapped key	Data consisting of an encrypted key that has been converted into a form that is usable by the TSIP driver by key injection. The wrapped key has been wrapped using an HUK, so the wrapped key of the same encrypted key will be a unique value on each device.
UFPK (User Factory Programming Key)	A keyring set by the user and used to generate an encrypted key from a user key during key injection. Not used on the device.
W-UFPK (Wrapped UFPK)	Key information generated by wrapping a UFPK using an HRK on the DLM server. The UFPK is decrypted using the HRK internally by the TSIP.
Hardware root key (HRK)	A shared encryption key that exists only inside the TSIP and in secure rooms within Renesas.
Hardware unique key (HUK)	A device-specific encryption key that is derived internally by the TSIP and used to protect key data.
Wrapping	In this document the term wrapping refers to the use of a UFPK to encrypt and MAC assignment as part of the process of generating an encrypted key. The TSIP driver does not accept plaintext input of user keys; they must be converted to an encrypted format (wrapped) before they can be input.
DLM server (Device Lifecycle Management server) <a href="https://dlm.renesas.com/keywrap/toEnglish">https://dlm.renesas.com/keywrap/toEnglish</a>	The key administration server at Renesas. Used to perform key wrapping (encrypting) of UFPKs.

## 2. Preparing the Sample Project

The sample project uses CK-RX65N Cloud Kit to establish a TLS connection to Microsoft Azure in order to demonstrate MQTT communication. A connection to Microsoft Azure IoT Hub is established via IoT Hub Device Provisioning Service (DPS). This section presents a guide explaining how to create the sample project used in the demo.

Information on making connections to the CK-RX65N board in order to run the sample project is provided below. To enable debugging and serial communication, connect the CK-RX65N board to the PC with two USB cables. To enable connection to the internet, connect the CK-RX65N board to a router with an Ethernet cable.

Note: Connection to Azure is also established via Ethernet in the case of Renesas Starter Kit+ for RX65N-2MB or RX72N Envision Kit.

The CK-RX65N board described in this document is the Ethernet version. If you are using the Cellular version of the CK-RX65N board, use mobile network connection by connecting the RYZ014A board (included in the kit) to the PMOD1 Connector of the CK-RX65N board.

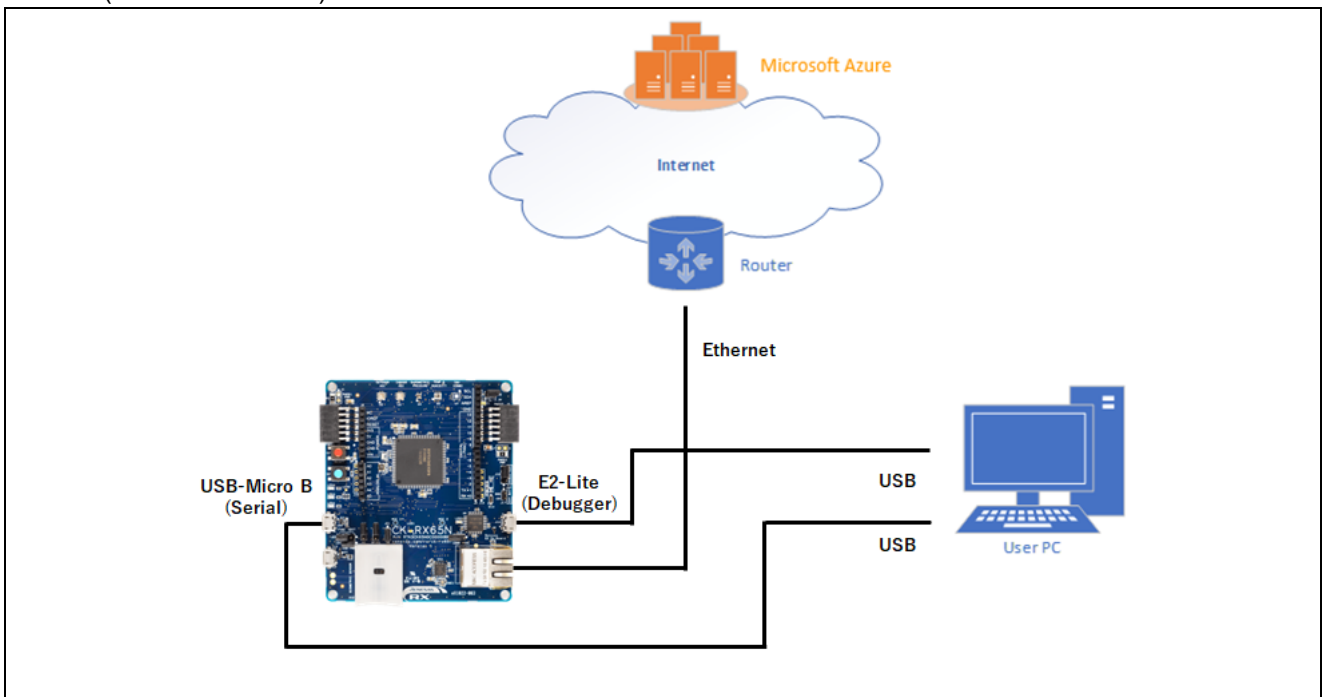


Figure 2-1 Sample Project Connections

The sample project is based on the Azure RTOS project. Azure RTOS provides IoT Libraries that bring together necessary source code for use with IoT devices. One of these is NetX Secure, a cryptographic library made use of by the sample project.

Next, we describe the software components of the sample project.



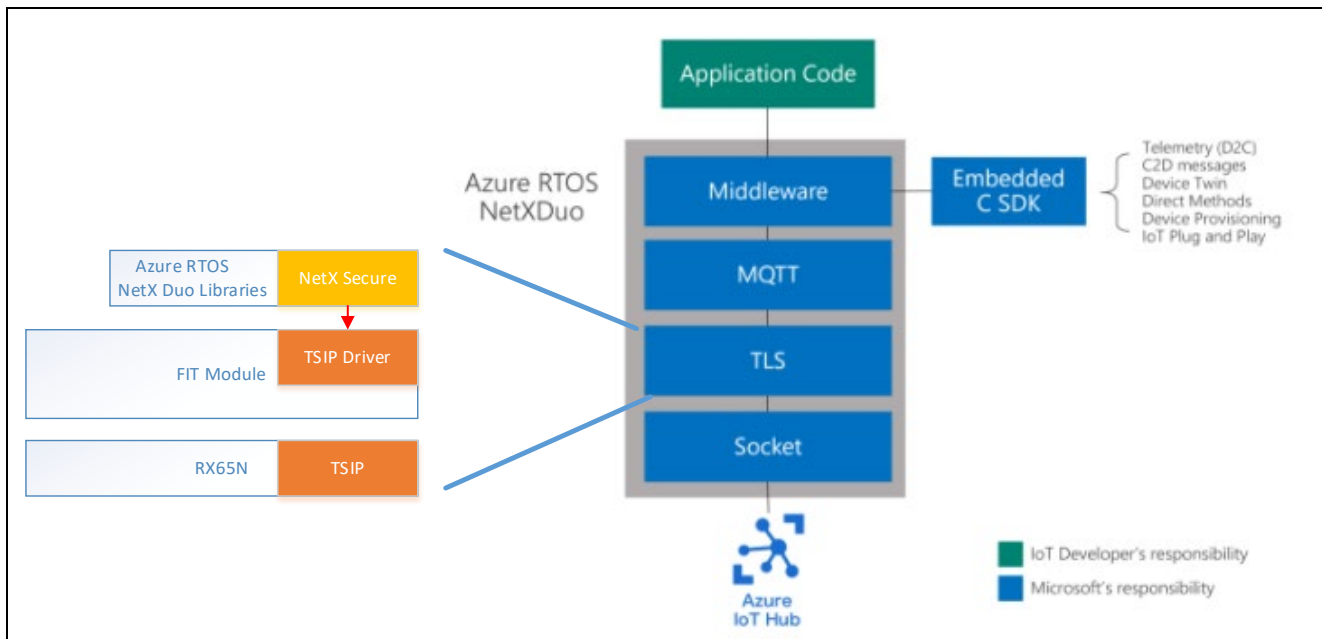


Figure 2-2 Sample Project Software Components

The sample project is based on version 6.2.1 of the Azure RTOS project for RX MCUs, which can be downloaded from the repository at the link shown below, and has further been modified in order to enable linkage with the TSIP. The modifications that were made can be checked from the differences between the sample project and its base project (downloadable at the following repository) by comparing them.

[https://github.com/renesas/azure-rtos/releases/tag/v6.2.1\\_rel-rx-1.3.0](https://github.com/renesas/azure-rtos/releases/tag/v6.2.1_rel-rx-1.3.0)

The procedure described in this section makes use of the following tools, which you will need to obtain before proceeding.

- Shell script (bash) execution environment (The procedure described in this document uses Cygwin.)
- OpenSSL
- Security Key Management Tool

## 2.1 Creating a Workspace

Launch e<sup>2</sup> studio and create a new workspace. Keep the names of the workspace and project files as short as possible. If the total length of the full file path exceeds 256 bytes, an error will occur when you build the project.

Example: Creating a workspace in location **C:\workspace**

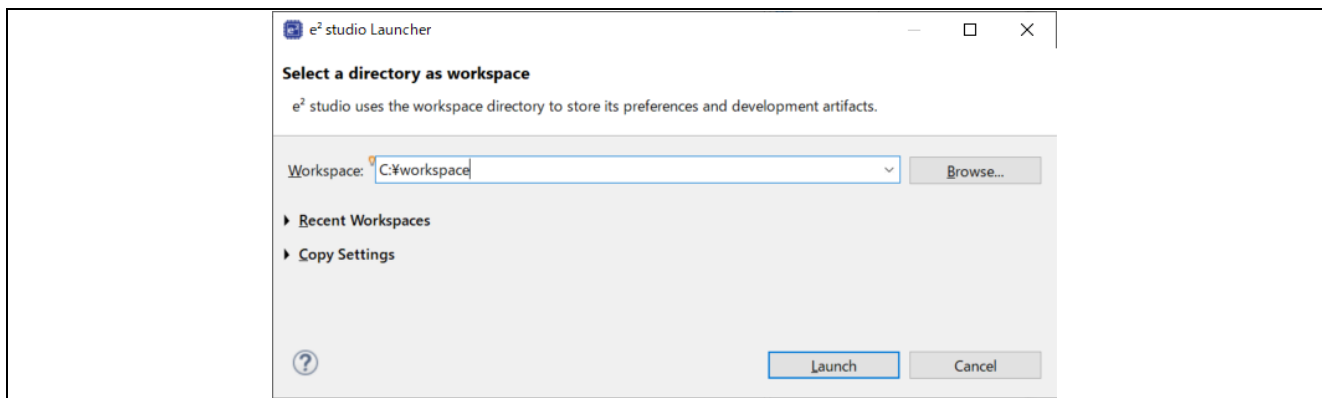


Figure 2-3 Select a directory as workspace Window

## 2.2 Downloading the Project

The sample project bundled with this document has the folder structure shown below. Copy these folders to the workspace folder created as described in 2.1, Creating a Workspace.

The first level of the folder hierarchy is shown below.

```
ckrx65n_ccrx
|--key_crt_sig_generator
| |--ca
| |--ca-sign-keypair-rsa2048
| |--client-rsa2048
| |--output
|--patch
| |--tsip_integration
|--sample_azure_iot_embedded_sdk
| |--libs
| |--src
```

Figure 2-4 Project Folder Structure

In addition to the source code of the sample project, the folders contain tools you will need to create the sample project. An outline of each folder is provided below.

Table 2-1 Contents of Sample Project

Folder Name	Description
key_crt_sig_generator	Contains tools and working folders used when generating keys and certificates used for encryption.
Patch	After the sample project has been generated, patch files used to modify the project are stored here. Since the relevant patches have already been applied to the sample project before it is imported, this folder will not be used in the procedure described here.
sample_azure_iot_embedded_sdk	The body of the sample project. You will import this folder by following the procedure described in the next section.

## 2.3 Importing the Project

After launching e<sup>2</sup> studio, from the menu bar select **File** → **Import...** → **General** → **Existing Projects into Workspace** to open the **Import Projects** dialog box. Click the

**Browse...** button to the right of **Select root directory:** and specify **sample\_azure\_iot\_embedded\_sdk** in the project folder **ckrx65n-ccrx**.

Confirm that **sample\_azure\_iot\_embedded\_sdk** appears under **Projects:**, check the box to select it, and click the **Finish** button.

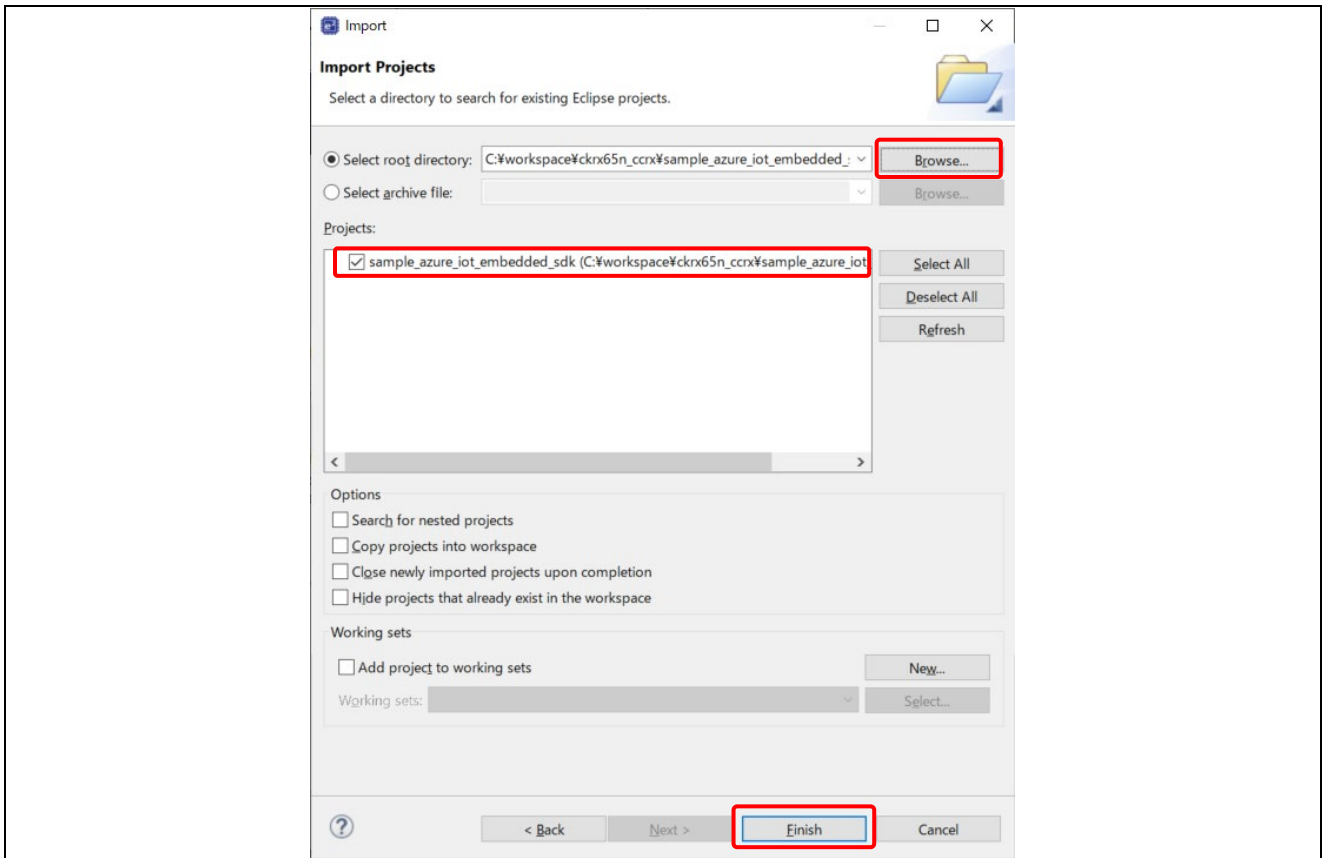


Figure 2-5 Import Projects Dialog Box

When the project has been imported successfully, **sample\_azure\_iot\_embedded\_sdk** appears in **Project Explorer** panel as shown below. If **Project Explorer** is not displayed, click **C/C++** at the upper right of the window to change the perspective and then select **Window** → **Show View** → **Project Explorer**.

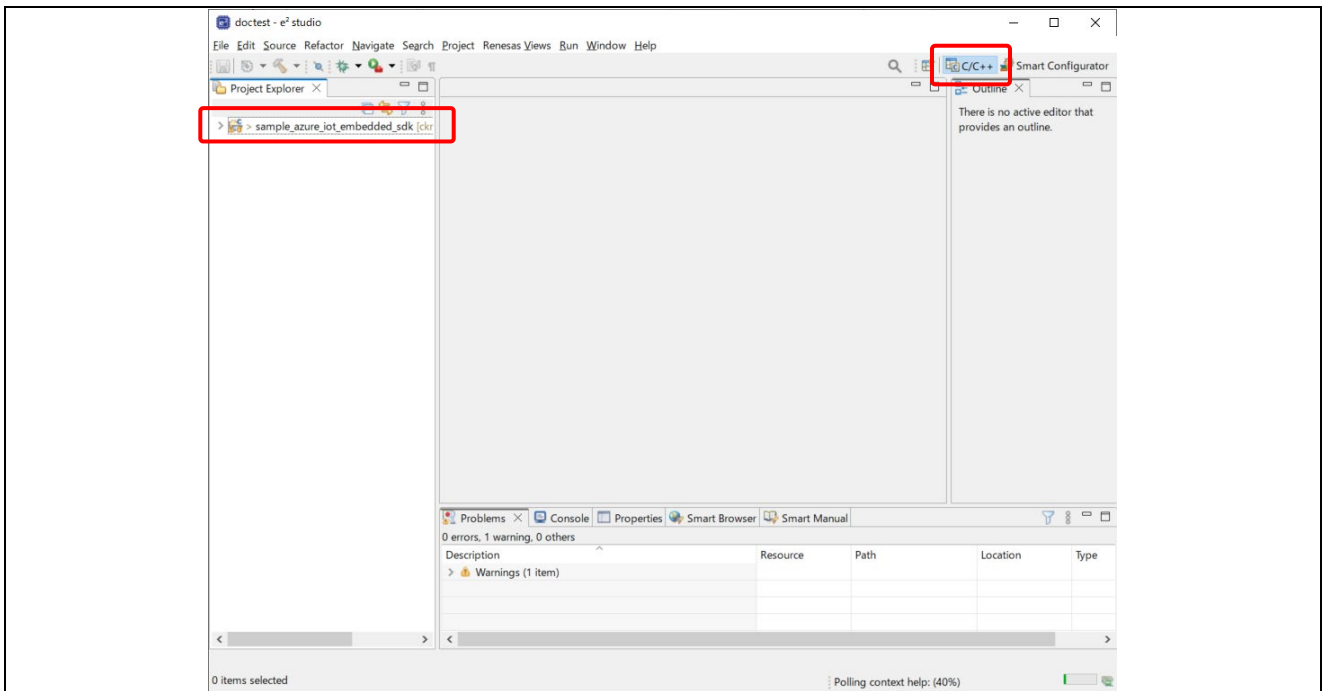


Figure 2-6 Window after Importing the Sample Project

## 2.4 Key and Certificate Preparation

You will need to register keying information and certificates listed in Table 2.2 in the sample program. This section explains how to obtain these keys and certificates as well as the procedures for converting them for use with the TSIP driver and for registering them in the project files. Follow the steps described below to generate keying information and certificates and register them in the project. The necessary information and how to obtain it are summarized in the table below.

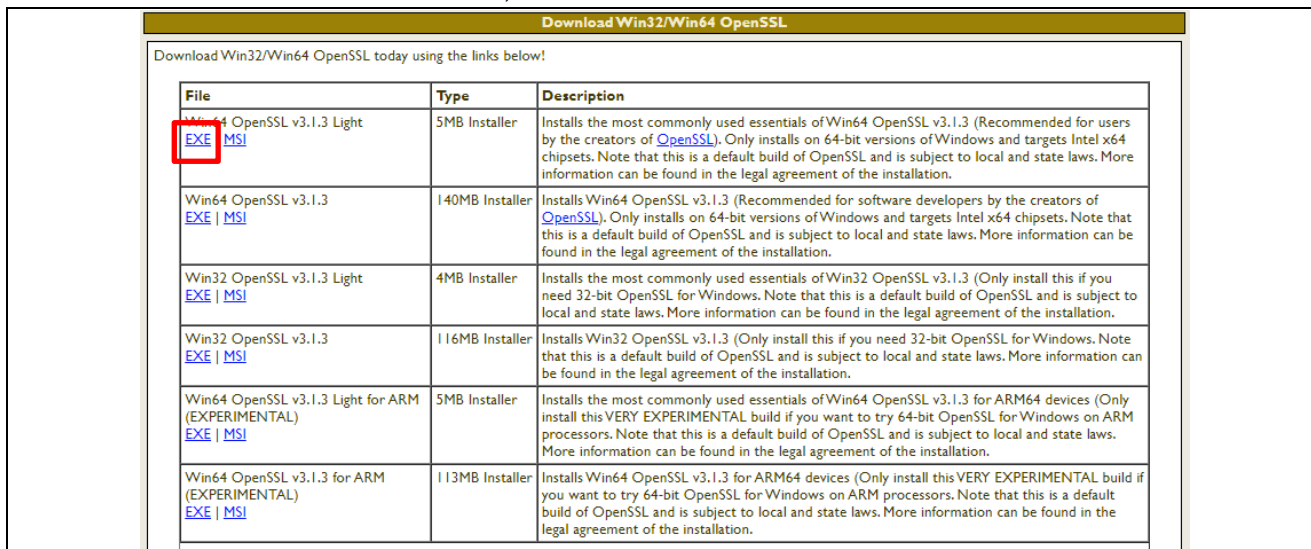
**Table 2-2 Methods for Obtaining Keys and Certificates for Use with Sample Project**

Key/Certificate	How to Obtain	Section
RSA root CA certificate	Download from dedicated link (DER format) or in web browser (PEM format).	2.4.2
RSA key pair	Created by user using OpenSSL tools or equivalent.	2.4.3
RSA client certificate	Created by user using OpenSSL tools or equivalent.	2.4.3
Key pair for root CA certificate signature generation and signature verification	Created by user using OpenSSL tools or equivalent.	2.4.4 2.4.5

### 2.4.1 Installing OpenSSL

OpenSSL is used to generate some keys and certificates. Follow the steps below to install OpenSSL.

1. Access the [download site](#) for the Win32 and Win64 versions of OpenSSL and download the installer that matches your OS. For the 64-bit version of Windows you would download v3.1.3 Light, shown below. If the version number shown is different, download the latest version available.



**Figure 2-7 OpenSSL Download Page**

2. Run the downloaded EXE (or MSI) file to install OpenSSL.
3. After the installation completes, in Windows select **System Properties** → **Environment Variables...** and add the OpenSSL install folder to the **Path** variable.  
64-bit version: C:\Program Files\OpenSSL-Win64\bin

## 2.4.2 Obtaining Root CA Certificate

The Azure cloud uses two types of root CA certificates: [Baltimore CyberTrust Root] and [DigiCert Global Root G2].

This sample project allows you to register up to three types of root CA certificates. The following steps will guide you through enrolling the two types of root CA certificates mentioned above.

Obtain an RSA root CA certificate and register it in the sample project. Either of the two procedures described below can be used to accomplish this; choose the one best suited to the environment you are using. Both procedures will enable you to obtain the same RSA root CA certificate.

1. Obtain from Microsoft Azure site
2. Obtain by exporting from a web browser (Microsoft Edge)

Note that the TLS certificates used by the Azure service are scheduled to change. Refer to the explanation on the page linked to below for details.

<https://techcommunity.microsoft.com/t5/internet-of-things-blog/azure-iot-tls-critical-changes-are-almost-here-and-why-you/ba-p/2393169>

The sample project uses the following two types of root CA certificates.

1. Baltimore Cyber Trust Root
2. DigiCert Global Root G2

### (1) Obtaining a Root CA Certificate

#### (a) Obtaining a Certificate from the Microsoft Azure Site

You can obtain a root CA certificate from the page linked to below.

<https://docs.microsoft.com/en-us/azure/security/fundamentals/tls-certificate-changes#what-is-changing>

Click the **Baltimore CyberTrust Root** and **DigiCert Global Root G2** link to download the RSA certificate.

**What changed?**

Prior to the change, most of the TLS certificates used by Azure services chained up to the following Root CA:

Common name of the CA	Thumbprint (SHA1)
<a href="#">Baltimore CyberTrust Root</a>	d4de20d05e66fc53fe1a50882c78db2852cae474

After the change, TLS certificates used by Azure services will chain up to one of the following Root CAs:

Common name of the CA	Thumbprint (SHA1)
<a href="#">DigiCert Global Root G2</a>	df3c24f9bfd666761b268073fe06d1cc8d4f82e4
<a href="#">DigiCert Global Root CA</a>	a8985d3a65e5e5c4b2d7d66d40c6dd2fb19c5436
<a href="#">Baltimore CyberTrust Root</a>	d4de20d05e66fc53fe1a50882c78db2852cae474
<a href="#">D-TRUST Root Class 3 CA 2 2009</a>	58e8abb0361533fb80f79b1b6d29d3ff8d5f00f0
<a href="#">Microsoft RSA Root Certificate Authority 2017</a>	73a5e64a3bff8316ff0edccc618a906e4eae4d74
<a href="#">Microsoft ECC Root Certificate Authority 2017</a>	999a64c37ff47d9fab95f14769891460eec4c3c5

**Figure 2-8 Obtaining an RSA Root Certificate**

The downloaded root CA certificate is a file in DER format with the following file name (and the extension CRT).

1. BaltimoreCyberTrustRoot.crt
2. DigiCertGlobalRootG2.crt

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

The root CA certificate must be in PEM format for use by the sample project. Therefore, follow the steps below to convert the downloaded root CA certificate from DER format to PEM format (extension: CER).

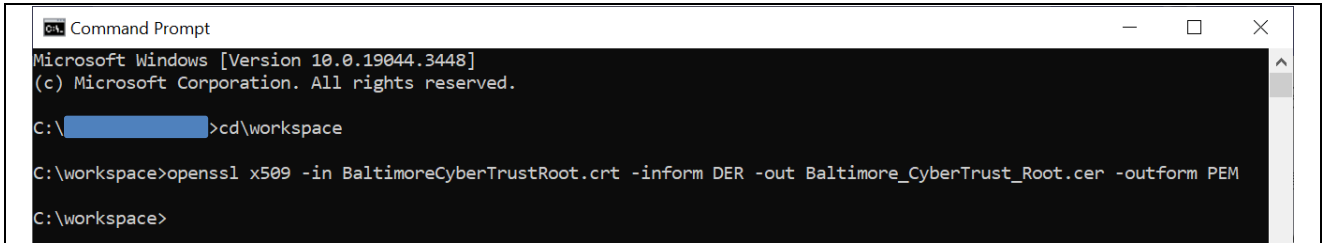
1. Copying the root CA certificate

Copy the downloaded **BaltimoreCyberTrustRoot.crt** and **DigiCertGlobalRootG2.crt** file to a working folder.

2. Converting the root CA certificate of the **Baltimore Cyber Trust Root** to PEM format

Open the Windows command prompt and enter the following OpenSSL command, shown in blue.

```
openssl x509 -in BaltimoreCyberTrustRoot.crt -inform DER -out Baltimore_CyberTrust_Root.cer -outform PEM
```



```
Command Prompt
Microsoft Windows [Version 10.0.19044.3448]
(c) Microsoft Corporation. All rights reserved.

C:\> cd\workspace

C:\workspace> openssl x509 -in BaltimoreCyberTrustRoot.crt -inform DER -out Baltimore_CyberTrust_Root.cer -outform PEM

C:\workspace>
```

**Figure 2-9 Conversion to PEM Format in OpenSSL(Baltimore)**

3. Converting the root CA certificate of the **DigiCert Global Root G2** to PEM format

Open the Windows command prompt and enter the following OpenSSL command, shown in blue.

```
openssl x509 -in DigiCertGlobalRootG2.crt -inform DER -out DigiCertGlobalRootG2.cer -outform PEM
```



```
Command Prompt
Microsoft Windows [Version 10.0.19044.3448]
(c) Microsoft Corporation. All rights reserved.

C:\> cd\workspace

C:\workspace> openssl x509 -in DigiCertGlobalRootG2.crt -inform DER -out DigiCertGlobalRootG2.cer -outform PEM

C:\workspace>
```

**Figure 2-10 Conversion to PEM Format in OpenSSL(DigiCert)**

4. When the conversion completes, two types of PEM format files will be created.

Baltimore\_CyberTrust\_Root.cer

DigiCertGlobalRootG2.cer

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

### (b) Using a Web Browser (Microsoft Edge) to Obtain a Certificate

The steps for using a web browser (Microsoft Edge) to download a root CA certificate in PEM format are as follows.

1. Click the ... (**Settings and more**) button in the upper right of the Microsoft Edge window and select **Settings** from the menu.

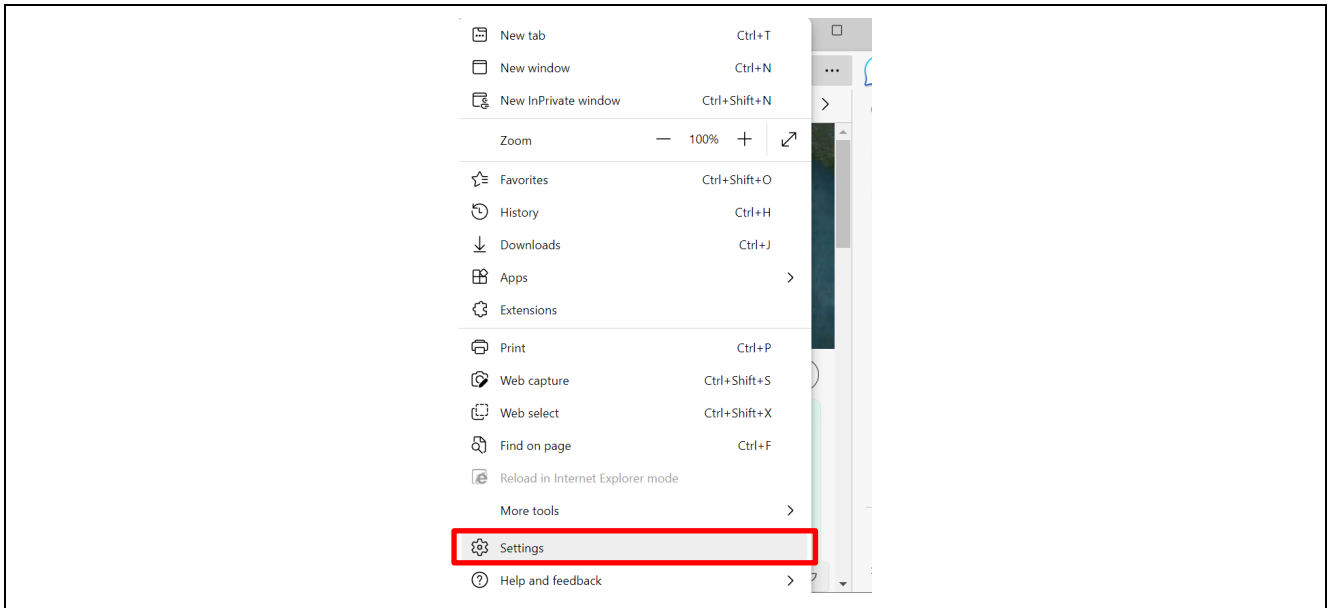


Figure 2-11 Microsoft Edge Settings Window

2. On the Microsoft Edge **Settings** menu, select **Privacy, search, and services**, and under **Security** click the icon to the right of **Manage certificates**.

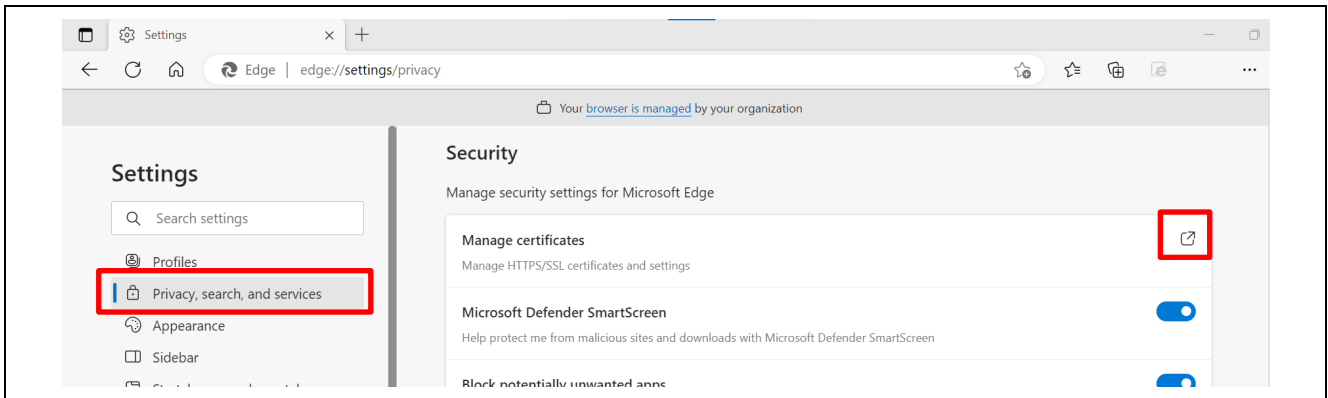


Figure 2-12 Microsoft Edge Settings Menu

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

3. On the **Certificates** dialog box, select the **Trusted Root Certification Authorities** tab, select the root CA certificate with **Baltimore CyberTrust Root** designated as both **Issued To** and **Issued By**, and click the **Export...** button. The **Welcome** window of the **Certificate Export Wizard** appears.

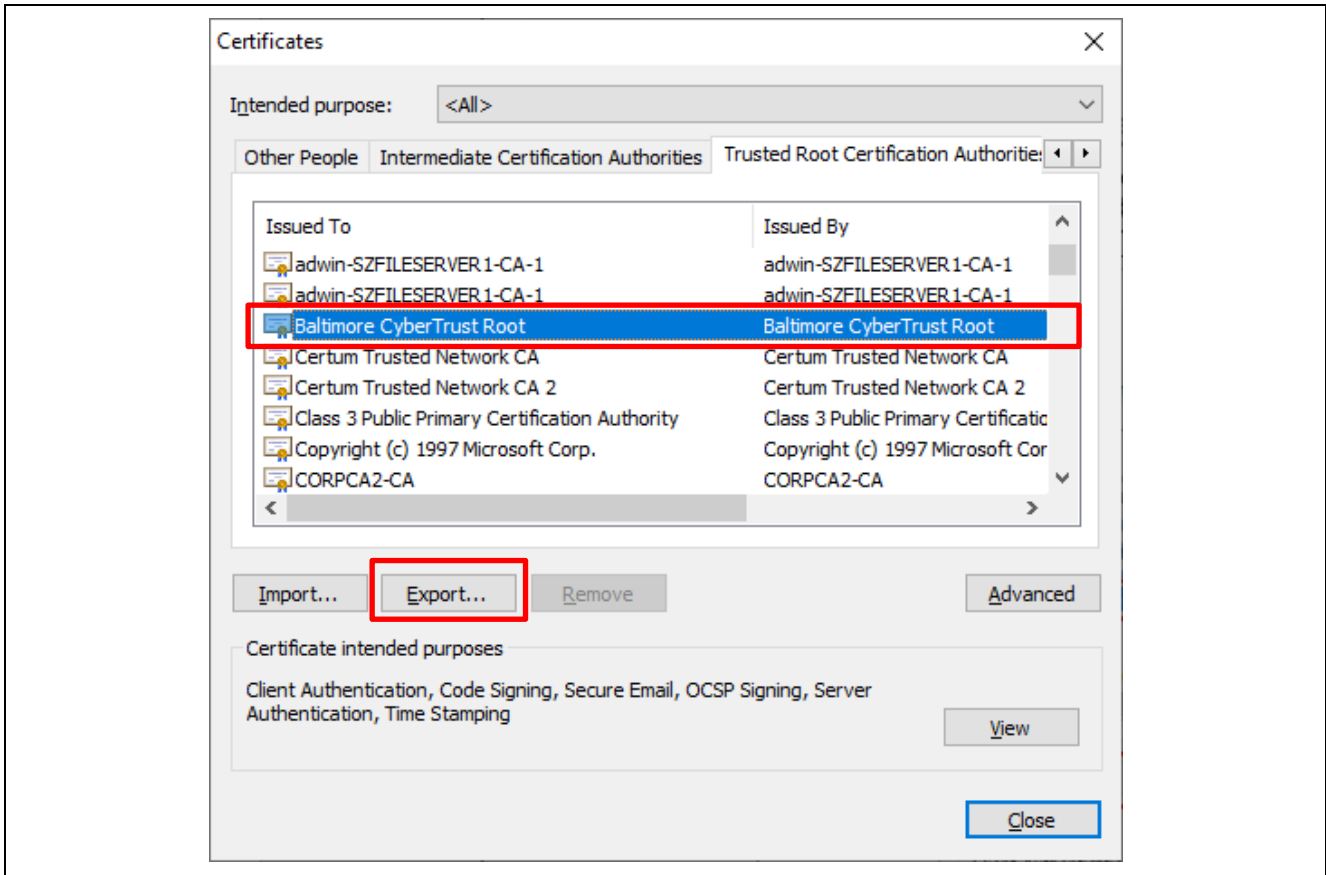


Figure 2-13 Certificates Dialog Box(Baltimore)



## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

- On the **Welcome** window of the **Certificate Export Wizard**, click the **Next** button. Then select **Base-64 encoded X.509 (.CER)** as the export file format and click the **Next** button.

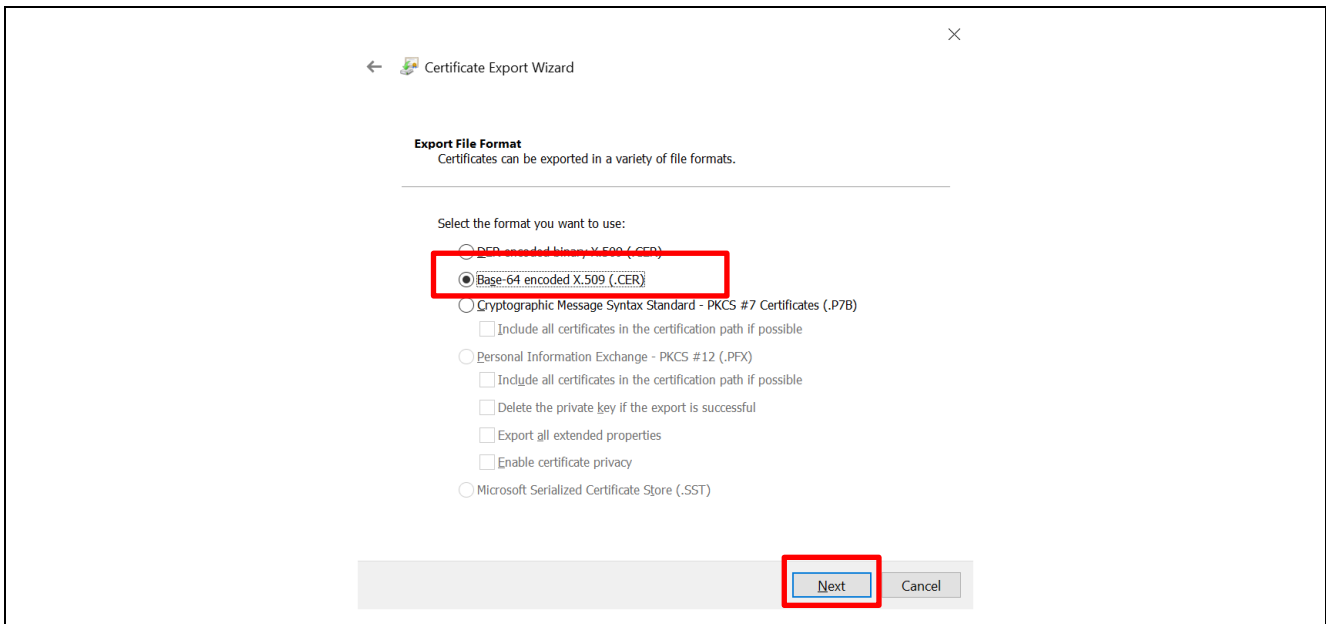


Figure 2-14 Specifying the Export File Format

- On the **File to Export** window, click the **Browse...** button, select the folder to export the file to, enter **Baltimore\_CyberTrust\_Root.cer** for **File name...**, and click the **Next** button.

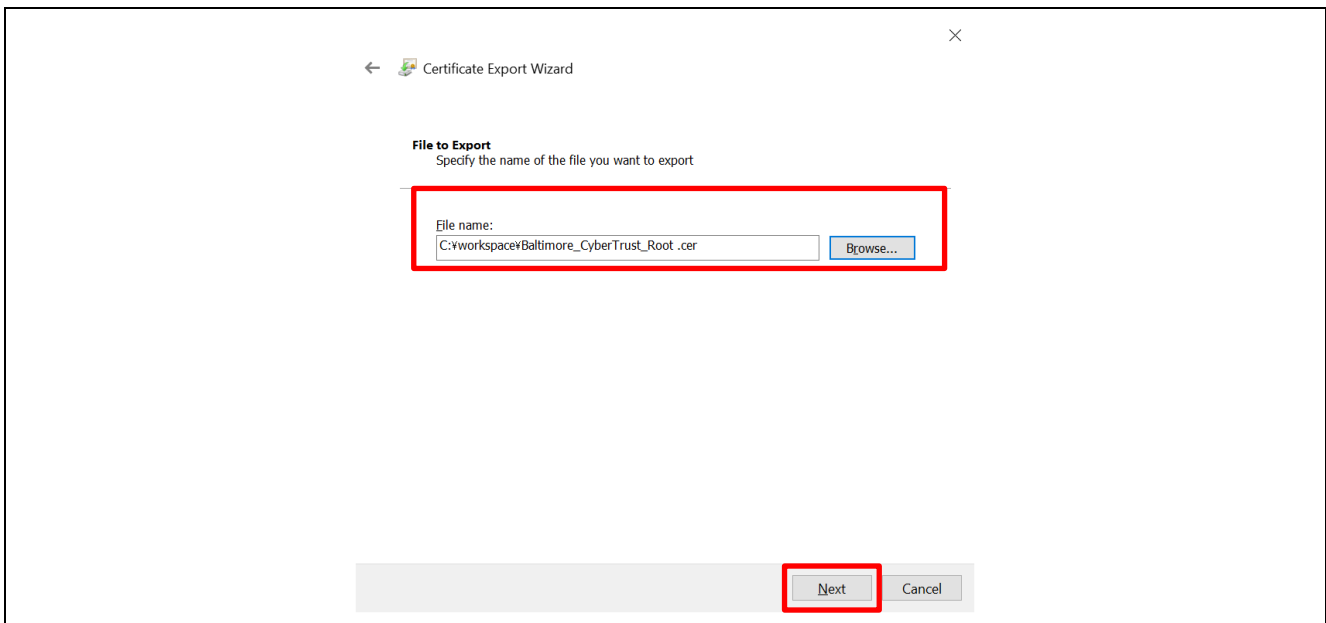
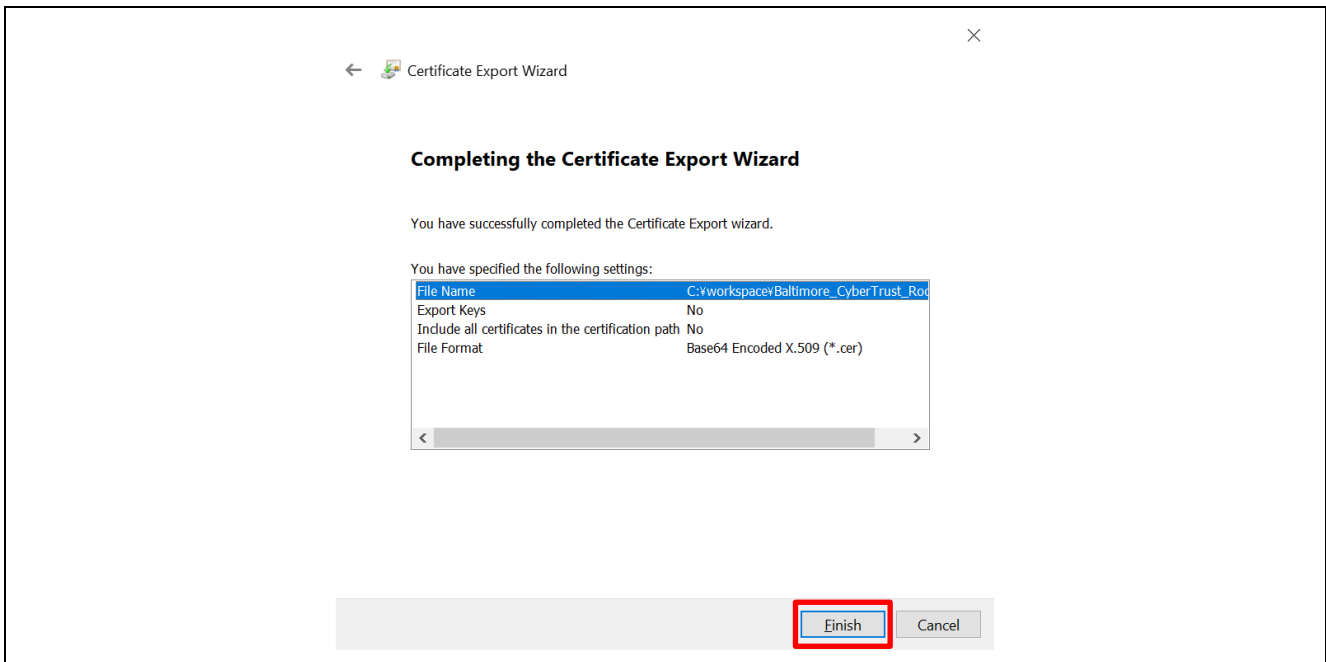


Figure 2-15 Specifying the Export File Name

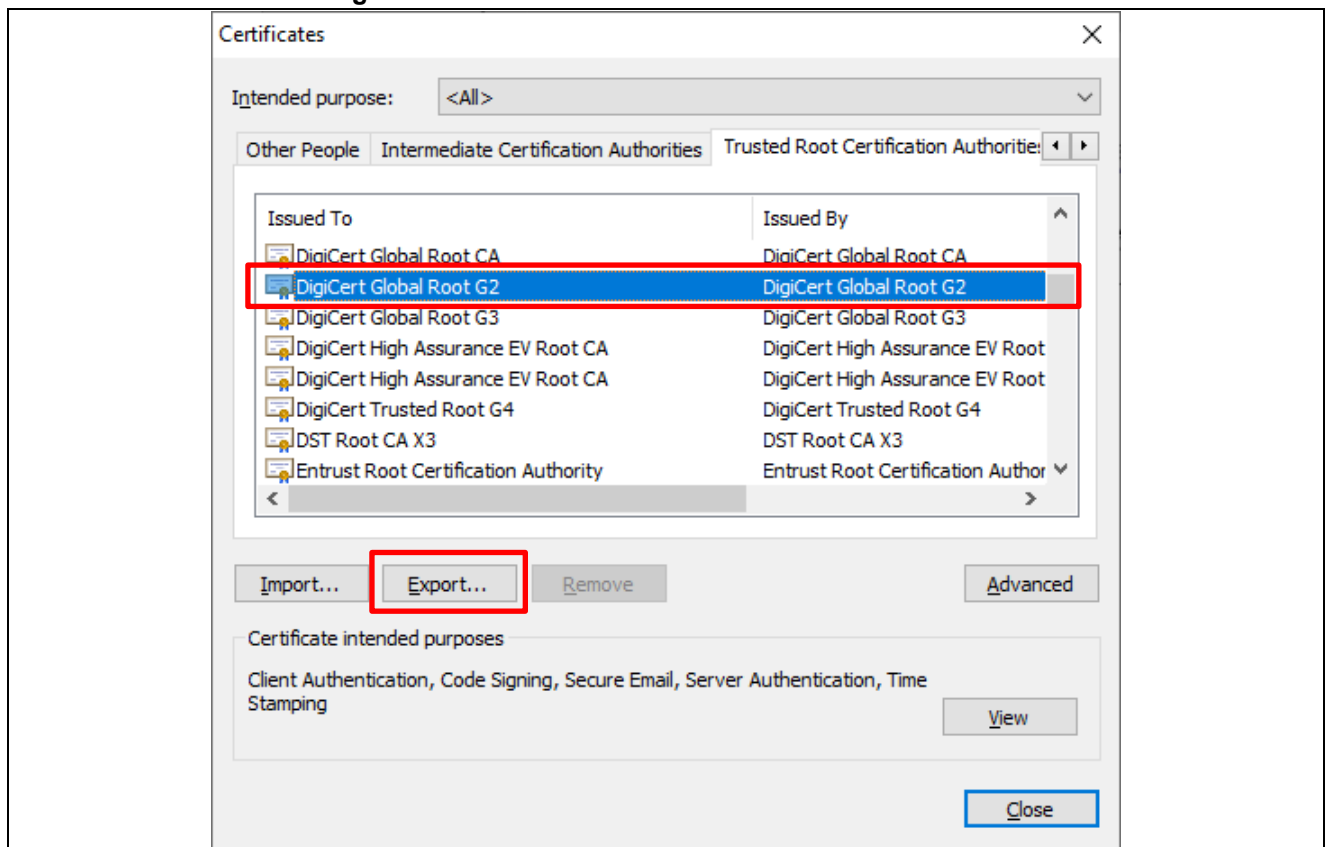
## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

- When the **Completing the Certificate Export Wizard** window appears, click the **Finish** button to export the root CA certificate.



**Figure 2-16** Completing the Certificate Export Wizard Window

- Export the root CA certificate of **DigiCert Global Root G2** by following steps 1 to 6. When exporting, enter the file name as **DigiCertGlobalRootG2.cer**.



**Figure 2-17** Certificates Dialog Box(DigiCert)

### (2) Placing the Newly Created Root CA Certificate in the Project Folder

Copy the root CA certificate exported as described in (1) to the **ca** folder in the **key\_cert\_sig\_generator** folder of the sample project (location shown in red). Do not change the file name as it will be used when running a script, as described below.

```
key_cert_sig_generator
|-- ca
|   |-- Baltimore_CyberTrust_Root.cer
|   |-- DigiCertGlobalRootG2.cer
|-- ca-sign-keypair-rsa2048
|-- client-rsa2048
|-- output
|--1_rsa2048_convertCert.sh
|--3_showkeyValues.sh
|--convertCert.sh
```

**Figure 2-18** Location of Root CA Certificate

### (3) About the root CA certificate used for the sample project

Connection to Azure IoT Hub or IoT Hub Device Provisioning Service (DPS) requires a root CA certificate. Set the root CA certificate in the sample project. The sample project provides the following three files, which can be used for registering the root CA certificate data:

- CyberTrust\_Root\_cert\_array\_1.txt
- CyberTrust\_Root\_cert\_array\_2.txt
- CyberTrust\_Root\_cert\_array\_3.txt

Now you have .cer files that you prepared in (2). The following sections describe how to convert these files into the above files and register the resulting files in the sample project as root CA certificate data. In the following sections, you will also generate a PSS-signed file that is necessary for verifying the root CA certificate.

In the project, you can register a maximum of three root CA certificates and PSS-signed files. Because the certificates are automatically validated by using the information registered in them, you do not need to set the certificate to be used.

**Note:** As of October 2023, Azure DPS only supports connections using Baltimore CyberTrust Root and IoT Hub only supports connections using DigiCert Global Root G2. Additionally, Azure DPS will be migrated to DigiCert Global Root G2. Please check Microsoft's Azure support information, etc. for the support status of root CA certificates in Azure.

## 2.4.3 Generating RSA Keys and Client Certificate

Generate an RSA key pair and client certificate, and register them in the sample project. You need to use a tool such as OpenSSL to accomplish this. When selecting a device authentication method (X.509 self-signed certificate) to perform TLS communication, you will need to create an RSA-2048 client certificate and client key pair (public key and private key).

### (1) Generating a Key Pair and Certificate

Follow the steps below to create an RSA key pair (for the client) and a client certificate (self-signed certificate).\*1

Note: 1. Do not change the file names of the key pair and certificate as they will be used when running a script, as described below.

1. Create a working folder for storing the newly generated key pair and certificate, then open the command prompt and change the active folder to the working folder.
2. Run the **openssl** command with the options shown below to create a key pair.  
`openssl genpkey -out device1-private.pem.key -algorithm RSA -pkeyopt rsa_keygen_bits:2048`  
The key pair is created with the file name **device1-private.pem.key**.

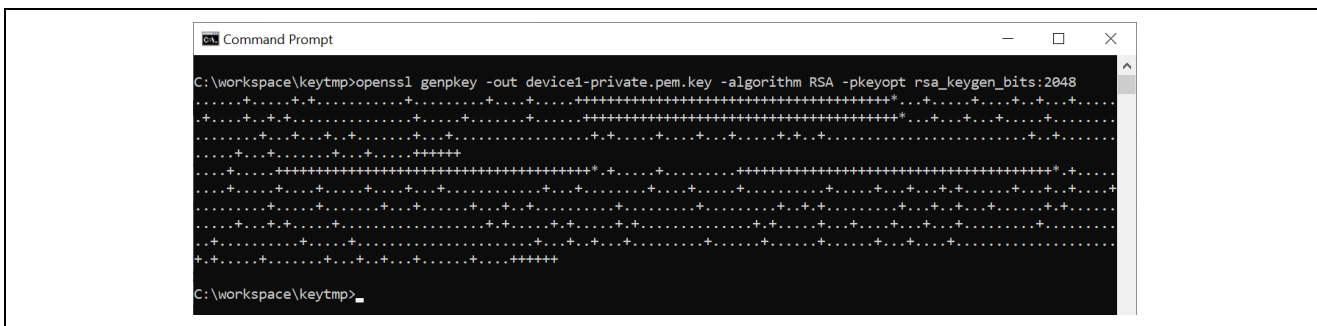


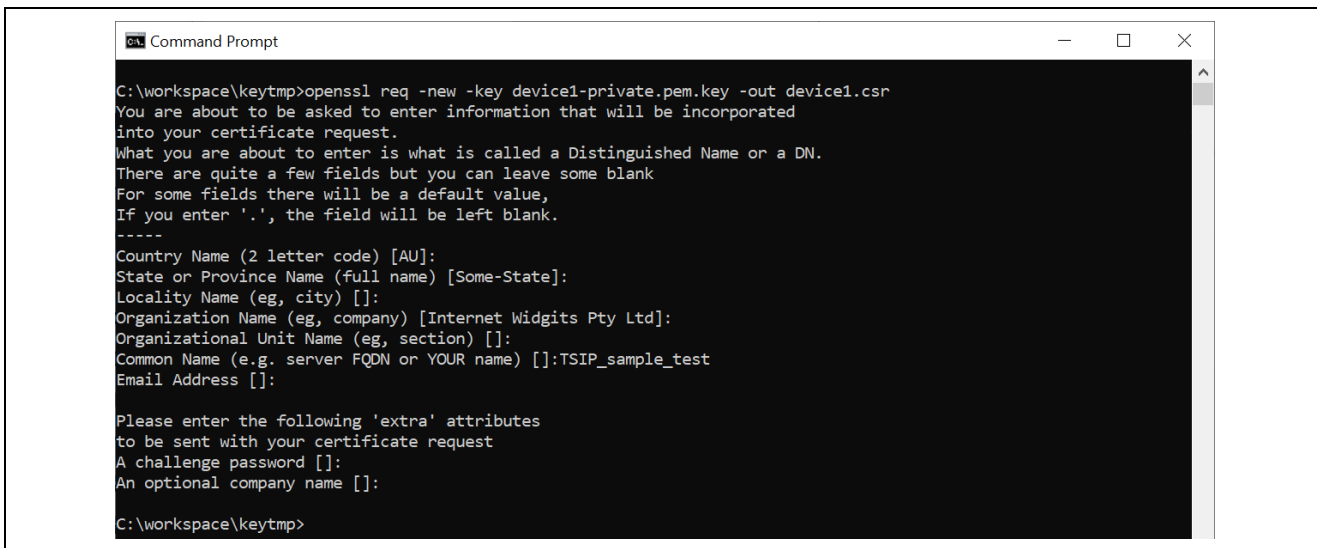
Figure 2-19 Creating a Key pair

3. Run the following command to create a certificate signing request (CSR).

`openssl req -new -key device1-private.pem.key -out device1.csr`

When you run the command, you are asked to enter parameters as shown below. You only need to enter a value of **TSIP\_sample\_test** for the item **Common Name**.\*1 For the other parameters, simply press the Enter key without inputting anything.

You are about to be asked to enter information that will be incorporated into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:  
State or Province Name (full name) [Some-State]:  
Locality Name (eg, city) []:  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:TSIP\_sample\_test  
Email Address []:  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:  
An optional company name []:

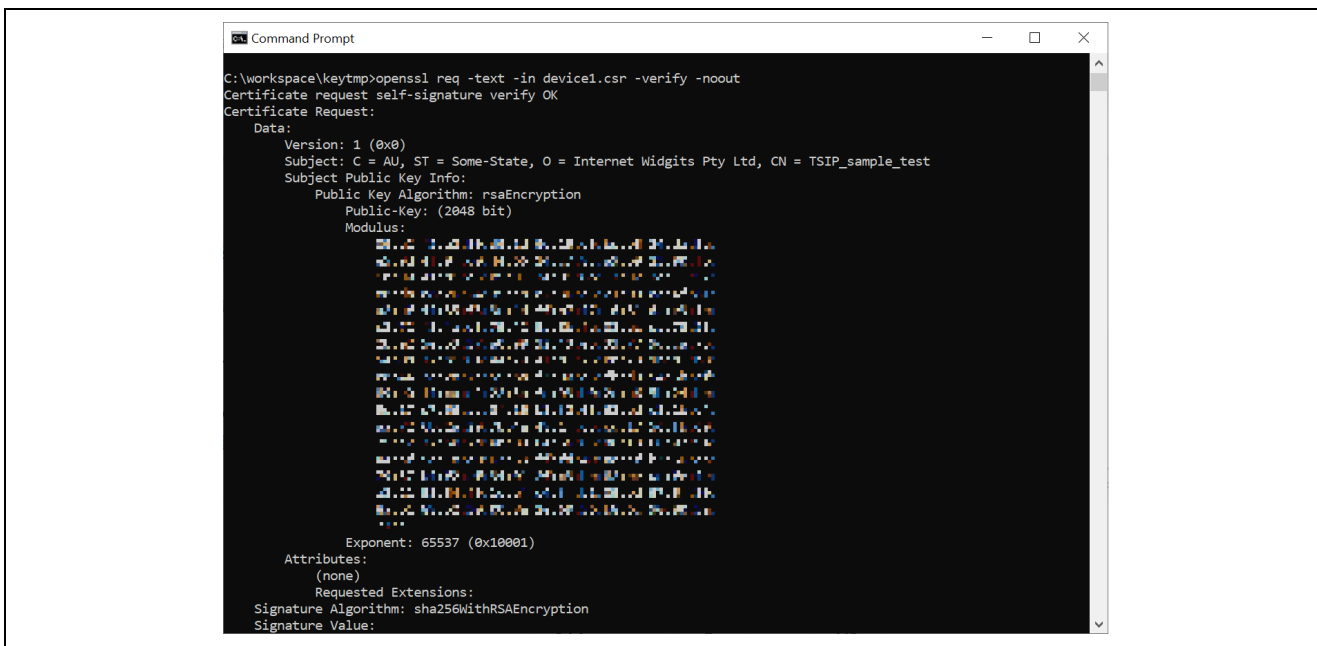


**Figure 2-20 Creating a Certificate Signing Request**

Note: 1. The **Common Name** parameter entered here is used as the registration ID (common name) for Azure IoT Hub DPS, as described below.

4. Next, enter the following command to confirm and verify the CSR.

```
openssl req -text -in device1.csr -verify -noout
```



**Figure 2-21 Confirming and Verifying the CSR**

5. Next, enter the following command to create a client certificate.

```
openssl x509 -req -days 365 -in device1.csr -signkey device1-private.pem.key -out device1-certificate.pem.crt
```

A client certificate is created with the file name **device1-certificate.pem.crt**.



**Figure 2-22 Creating a Self-Signed Certificate**

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

This newly generated client certificate will also be used to register on Azure IoT Hub Device Provisioning Service (DPS).

### (2) Placing the Newly Created Key and Certificate in the Project Folder

Copy the key pair file **device1-private.pem.key** and client certificate file **device1-certificate.pem.crt** created as described in (1) to the **client-rsa2048** folder in the **key\_cert\_sig\_generator** folder of the sample project (location shown in red). Do not change the file names as they will be used when running a script, as described below.

```
key_cert_sig_generator
|-- ca
|   |-- Baltimore_CyberTrust_Root.cer
|   |-- DigiCertGlobalRootG2.cer
|-- ca-sign-keypair-rsa2048
|-- client-rsa2048
|   |-- device1-certificate.pem.crt
|   |-- device1-private.pem.key
|-- output
|-- 1_rsa2048_convertCert.sh
|-- 3_showkeyValues.sh
|-- convertCert.sh
```

Figure 2-23 Location of Private Key and Certificate

### 2.4.4 Root CA Certificate Signature Generation and Certificate File Format Conversion

Create a root CA certificate and client certificate, and then register them in the sample project (source code). Certificates used with TLS are generally provided in PEM format, but the TSIP driver used by the sample project requires that certificates be converted from PEM format to DER format. Follow the steps described below to convert the certificates. A script file is provided in the **key\_cert\_sig\_generator** project folder to perform the necessary conversions. Run the script file as a shell script (bash). The example below uses Cygwin to run the script.

Before running the script, follow the steps described in sections 2.4.2 and 2.4.3 to ensure that the key and certificate files are located in the designated folders.

#### (1) Converting the RSA Certificate

Run the script to convert the RSA root CA certificate and client certificate to DER format. The script generates an RSA 2048-bit key pair for generating and verifying the root CA certificate signature and then uses the private key from the generated key pair to generate a signature for the root CA certificate. After conversion to DER format, each certificate is converted to the C language array format to allow it to be registered in the source code of the sample project.

Launch Cygwin and change the active directory to the **key\_cert\_sig\_generator** folder of the sample project. Then enter the following command to run the script.

`./1_rsa2048_convertCert.sh`

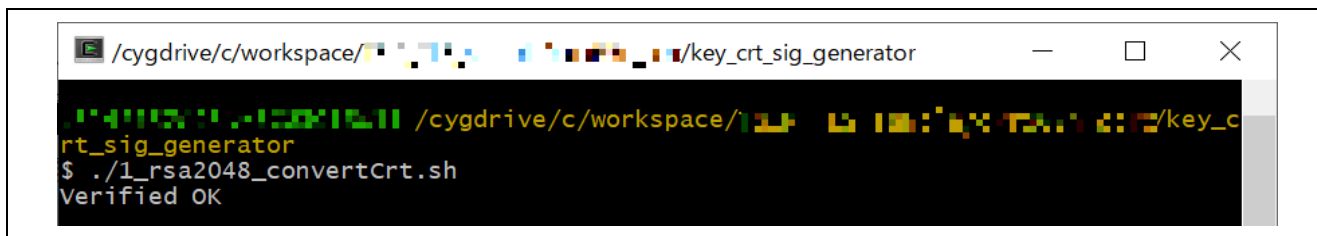


Figure 2-24 Running the Script

### (2) Registering the Converted Files in the Project

After the script runs, the following eight files will have been created in the **key\_cert\_sig\_generator/output** folder.

- Baltimore\_CyberTrust\_Root\_crt\_array.txt
- Baltimore\_CyberTrust\_Root\_sig\_array.txt
- DigiCertGlobalRootG2\_crt\_array.txt
- DigiCertGlobalRootG2\_sig\_array.txt
- client\_rsa2048\_crt\_array.txt
- Baltimore\_CyberTrust\_Root\_crt.der
- Baltimore\_CyberTrust\_Root\_sig.sig
- client\_rsa2048\_crt.der

Of these, the files listed in **red** are used by the sample project. These files contain generated binary data in C language uint8\_t array format.

Rename the four files that are related to the root CA certificate as shown in the following table.

**Table 2-3 Renaming the Root CA Certificate Files**

Before the Renaming	After the Renaming
Baltimore_CyberTrust_Root_crt_array.txt	CyberTrust_Root_crt_array_1.txt
Baltimore_CyberTrust_Root_sig_array.txt	CyberTrust_Root_sig_array_1.txt
DigiCertGlobalRootG2_crt_array.txt	CyberTrust_Root_crt_array_2.txt
DigiCertGlobalRootG2_sig_array.txt	CyberTrust_Root_sig_array_2.txt

Save the above four renamed files and the client\_rsa2048\_crt\_array.txt file in the following folder in the sample project:

**sample\_azure\_iot\_embedded\_sdk/src/userdata\_tsip**

Overwrite the files existing in the folder with the new files.

In addition, during the above certificate generation process, the key pair and public key file for signature verification of the root CA certificate shown below are generated in the **key\_cert\_sig\_generator /ca-sign-keypair-rsa2048** folder.

The key pair file (**rsa2048-private.pem**) is also used in the process in Section 2.4.5.

1. rsa2048-private.pem(key pair file)
2. rsa2048-public.pem(public key file)

### 2.4.5 Key Wrapping and Registration in the Project

Register in the sample project (source code) a key pair for verifying the signature of the generated in section 2.4.3 and 2.4.4, root certificate and client certificate keying information. The TSIP driver does not accept input of user keys in plaintext, so the keys must be “wrapped” to convert them to a format that will be accepted by the TSIP driver.

As with certificates, keys used for TLS are generally provided in PEM format. For use by the TSIP driver, the keying data must be extracted from the PEM format key file. After this, wrap it using the [Renesas DLM server](#) (Renesas Key Wrap service) and the Security Key Management Tool.

OpenPGP encryption is required in order to exchange keying data with the Renesas DLM server. The procedure described below uses Gpg4win (Kleopatra) for OpenPGP encryption. The following is an overview of the procedure.

1. Use the Security Key Management Tool to create a plaintext UFPK.
2. Use Kleopatra to apply PGP encryption to the UFPK (to enable exchanges of data associated with key wrapping).
3. Use the Renesas Key Wrap service to send the PGP-encrypted UFPK to Renesas.
4. Receive an encrypted UFPK (PGP-encrypted for transmission) from Renesas.
5. Use Kleopatra to decrypt the PGP encryption and obtain the encrypted UFPK (W-UFPK).
6. Use the Security Key Management Tool to verify the signature of the root certificate and wrap the client certificate key pair information using the UFPK and W-UFPK.
7. Register the wrapped encrypted key files in the source code.

Detailed steps are given below.



### 2.4.5.1 Creating a UFPK and W-UFPK

Generate a random User Factory Programming Key (UFPK) and upload it to the DLM server to generate a W-UFPK (a UFPK wrapped by the Renesas Key Wrapping service). The UFPK is used to wrap the public key used for signature verification.

You can use the Security Key Management Tool to create a UFPK file\*<sup>1</sup> in a format that will be accepted by the DLM server.

Note: 1 The procedure for generating a UFPK for wrapping and an encrypted UFPK (W-UFPK) is described in this section.

However, the keying information generated as described here is for use with the sample project and cannot be used in an actual product. For use in mass production, etc., it is necessary to generate a unique key. This process is covered in detail in a separate application note. This documentation is available to customers utilizing, or considering utilizing, Renesas MCUs.

Please contact your Renesas Electronics sales office for details. <https://www.renesas.com/contact/>

#### (1) Registration on the Renesas Key Wrap Service

To perform encryption on the DLM server, it is necessary at the time of the first transaction only to perform an OpenPGP key exchange with the Renesas DLM server. Log in on the page linked to below to perform the initial registration.

[Login screen of Key Wrap service \(renesas.com\)](#)

For details, refer to the Key Wrap service FAQ linked to below.

[KeyWrap Service Operation Manual.pdf \(renesas.com\)](#)

Next, use OpenPGP to generate a PGP key pair. The example procedure described in this document uses Gpg4win and Kleopatra. The steps for installing and using Gpg4win are described in section 8, Annex [Appendix], of the Key Wrap service manual linked to above. Follow the instructions to install Gpg4win.

### (2) Creating an OpenPGP Key Pair and Performing an OpenPGP Key Exchange with the Renesas DLM Server

You need to create a key pair in Kleopatra before you can perform an OpenPGP key exchange with the Renesas DLM server. This procedure is necessary only for the first transaction. Launch Kleopatra and click the **New Key Pair** button on the **Certificates** tab. When the **Create OpenPGP Certificate** dialog box appears, click the **Advanced Settings...** button.

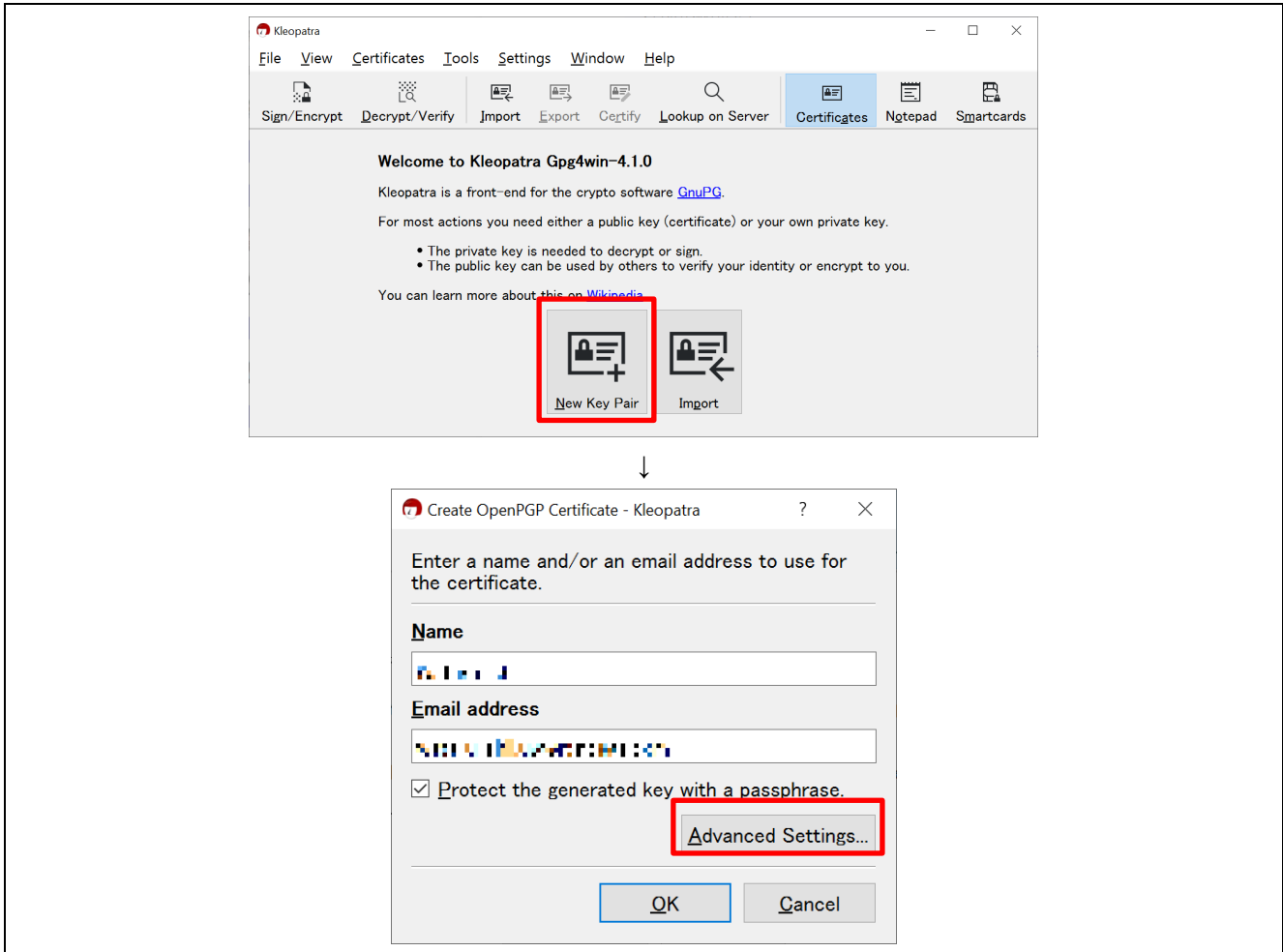


Figure 2-25 Generating a New Key Pair

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

On the **Advanced Settings** dialog box, enter the settings shown to specify **RSA** and **4,096 bits**, then click the **OK** button. The Renesas DLM server only supports exchange of RSA keys.

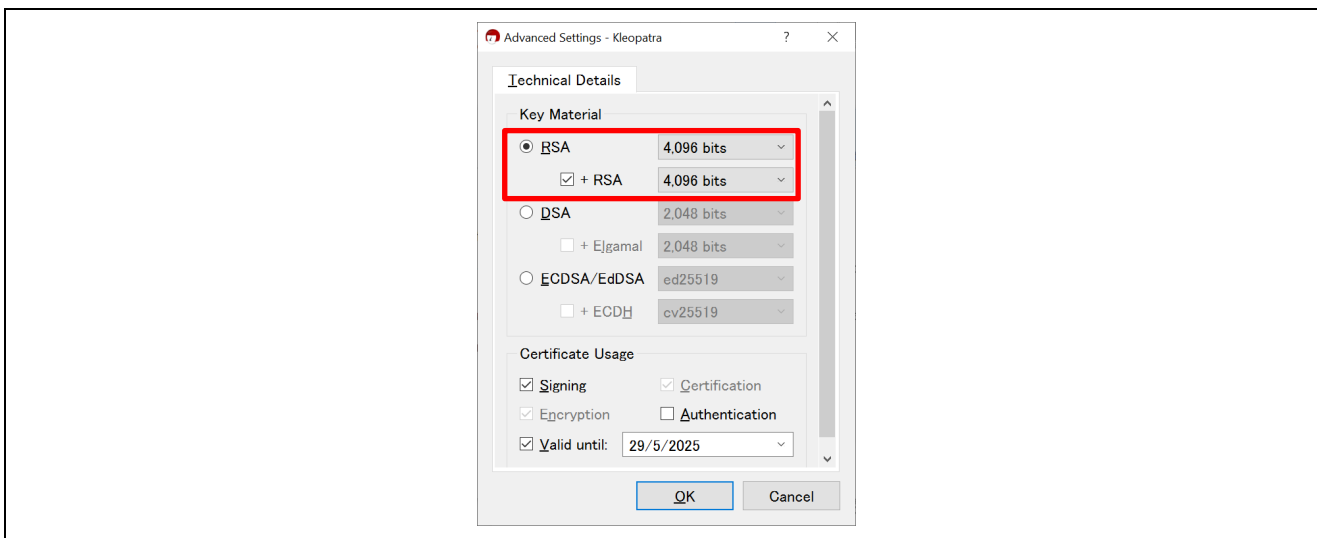


Figure 2-26 Advanced Settings Dialog Box

On the dialog box that appears after clicking the **New Key Pair** button, enter values for **Name** and **Email address**, then click the **OK** button. You can also check the box next to **Protect the generated key with a passphrase** to include a passphrase for additional security. If you use this setting, make sure you do not forget the passphrase.

After the key pair has been created, the registered key pair information appears in the Kleopatra window, as shown below. Select the newly registered key pair and click the **Export** button to output the OpenPGP public key. The file name will have the extension ASC.

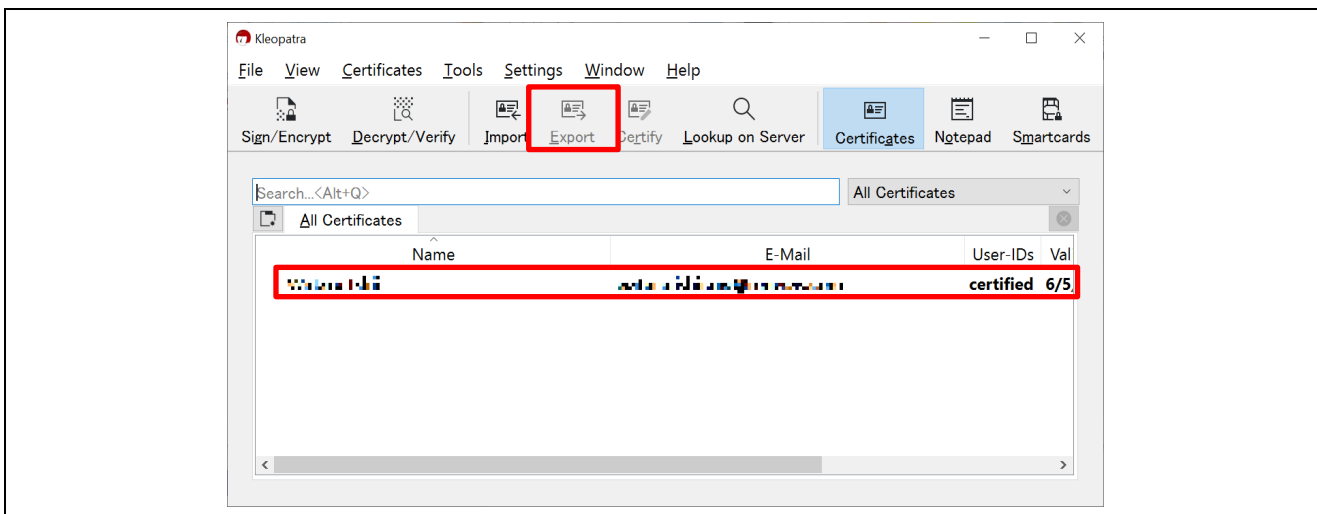


Figure 2-27 Outputting Your Own PGP Public Key

On the Renesas Key Wrap service website, click **PGP key exchange** and register the newly created OpenPGP public key. If the registration is successful, you will receive an email containing the Renesas PGP public key.

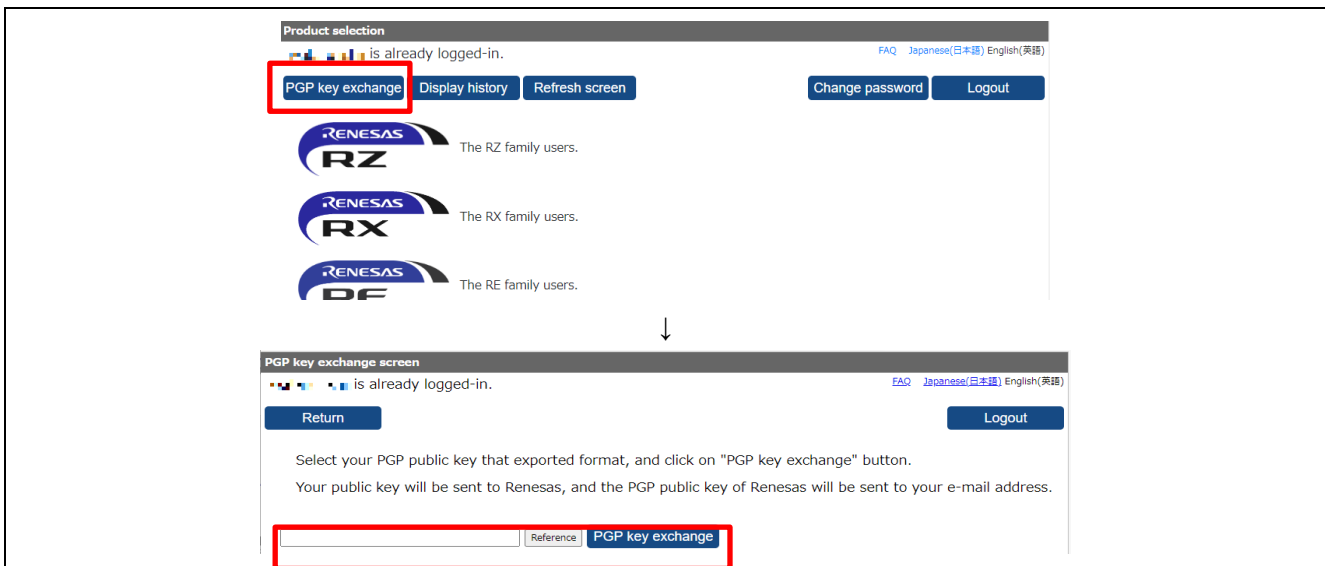


Figure 2-28 PGP Public Key Exchange

(3) Registering the Renesas OpenPGP Public Key

The Renesas PGP public key is used by the DLM server to decrypt PGP encrypted keys. Register the Renesas PGP public key you received by email in Kleopatra. To do this, click the **Import** button on the Kleopatra toolbar.

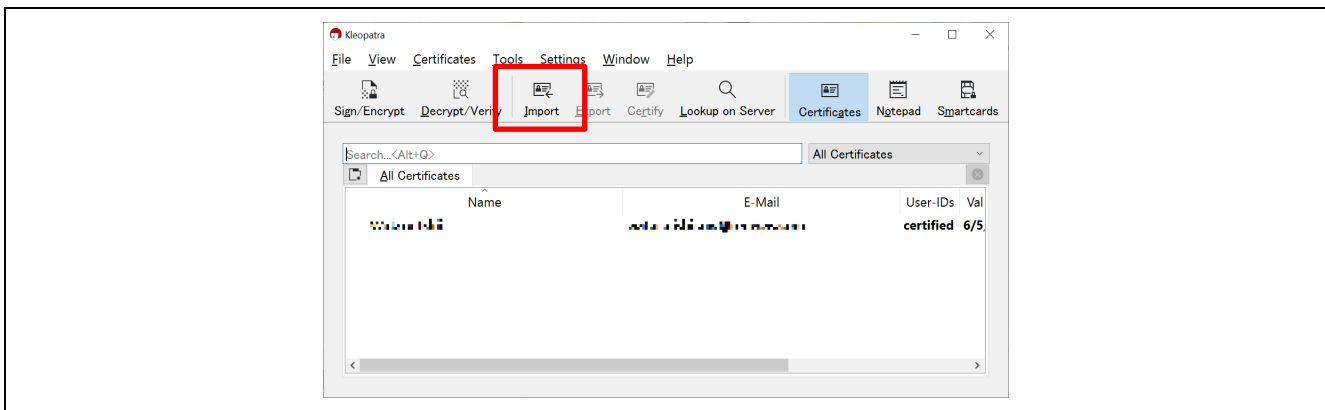


Figure 2-29 Importing the Renesas PGP Public Key

The **Select Certificate File** window appears to allow you to select the file to be imported. Select **Any files (\*)** as the extension, specify the **keywrap-pub.key** PGP public key file sent to you by Renesas, and click the **Open** button.

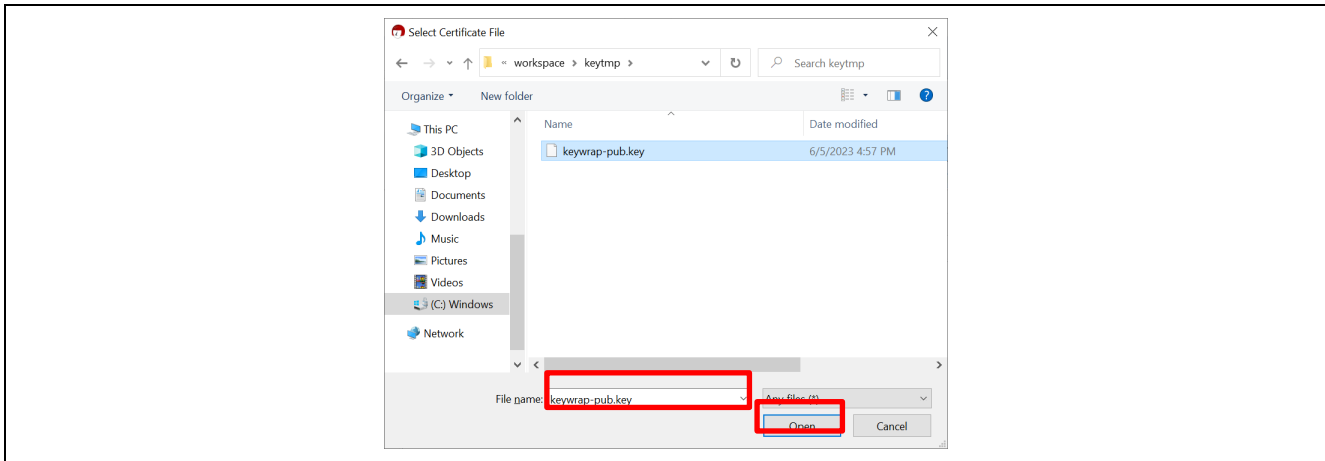


Figure 2-30 Select Certificate File Window

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

When the import completes, click the **OK** button to close the **Certificate Import Result** window. If a window for signature and encryption confirmation appears during the import process, select the previously registered key pair.

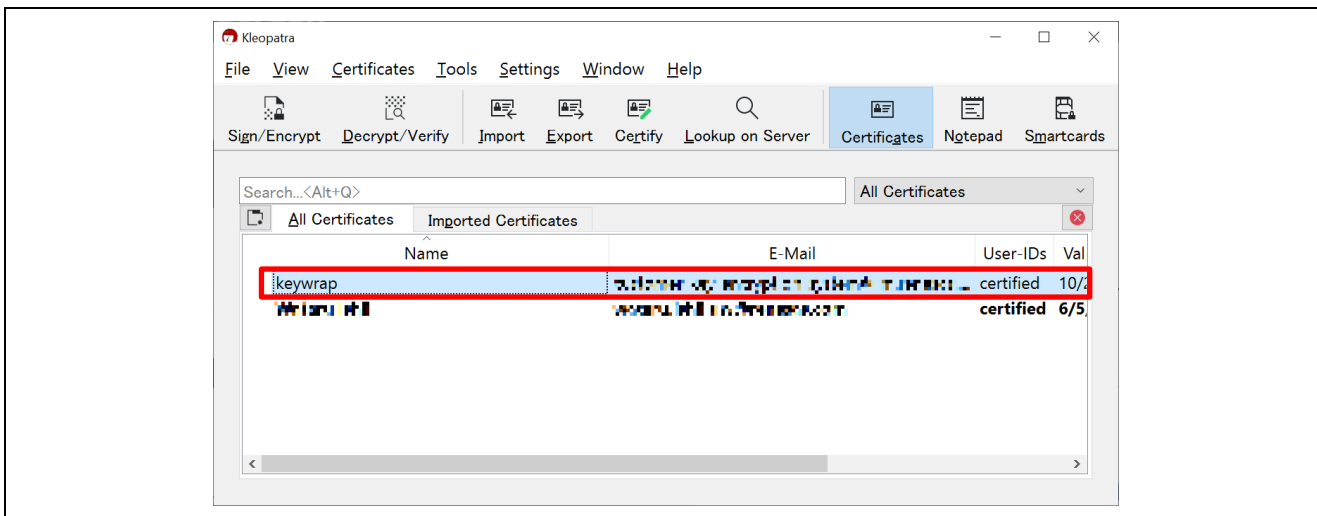


Figure 2-31 Renesas PGP Public Key Import Status

### (4) Generating a UFPK (Plaintext) File

Use the Security Key Management Tool to create a plaintext UFPK file.

Download the latest version of the Security Key Management Tool from the **Downloads** page at the link shown below. After downloading the relevant .zip file, unzip it, and then run the unzipped .exe file to install the software.

[Security Key Management Tool | Renesas](#)

After installing the Security Key Management Tool, start it, click the **Overview** tab, and then select **RX Family, TSIP**.

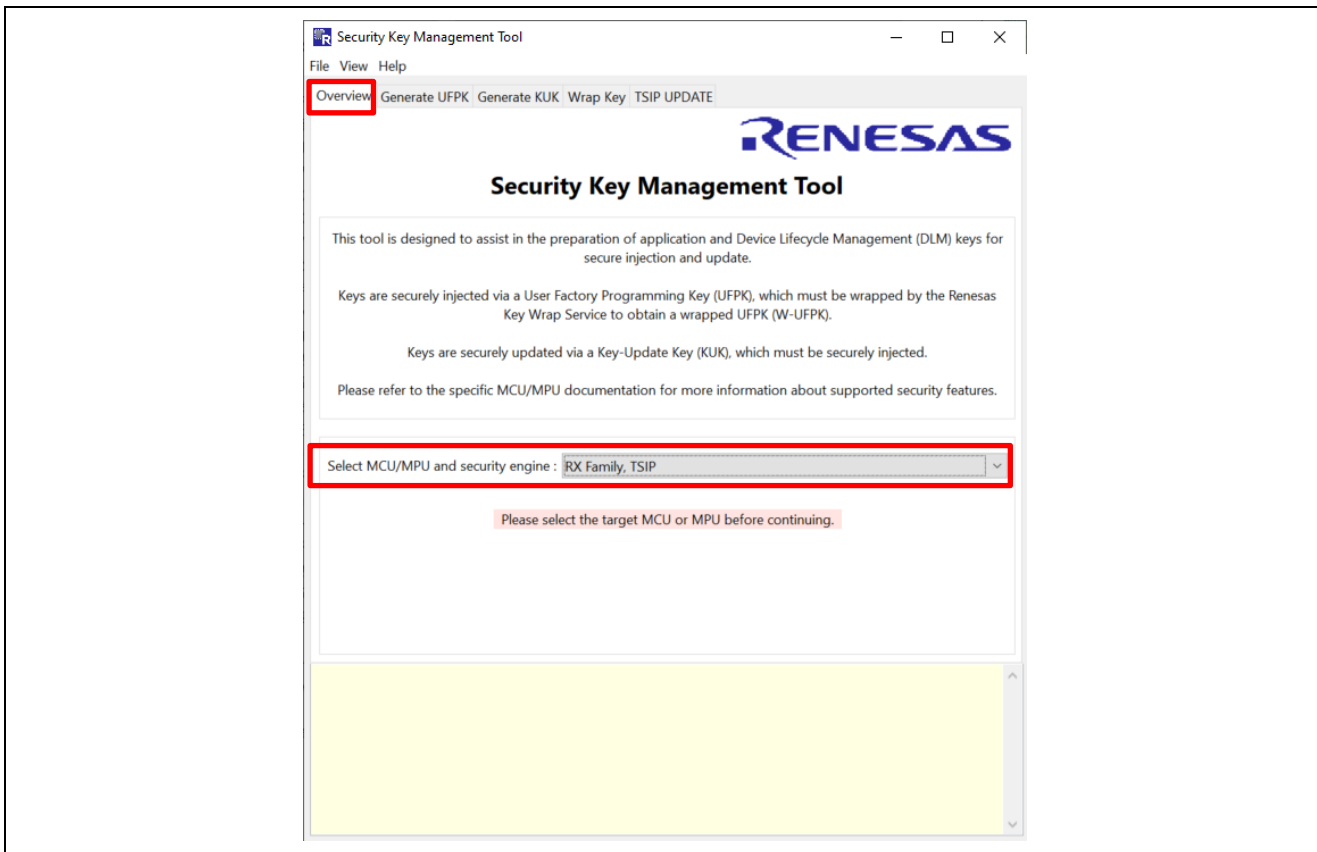


Figure 2-32 Selecting the MCU

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

In the Security Key Management Tool app, click the **Generate UFPK** tab, and then select **Generate random value**. In the **Output file (.key):** field, enter the name of the UFPK file to be output, with its full path name. In this example, **sample.key** is used as the file name. After entering the path and file name as shown in the following figure, click the **Generate UFPK key file** button. The UFPK file is output to the specified path.

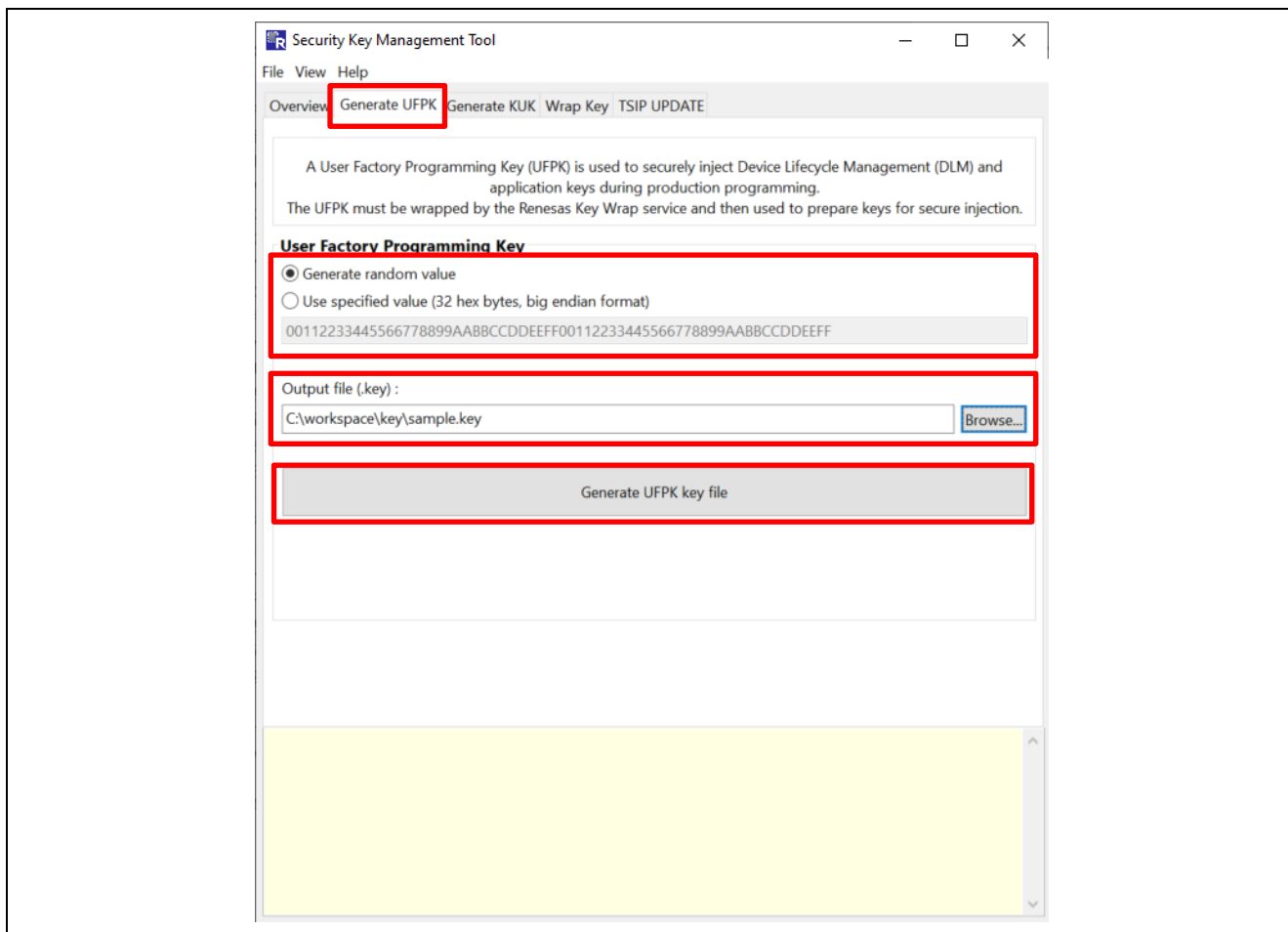


Figure 2-33 Generating a UFPK

### (5) PGP Encrypting the UFPK

Use Kleopatra to PGP encrypt the UFPK (**sample.key**) created as described in (4) using the previously created OpenPGP key pair and the Renesas PGP public key. Click the **Sign/Encrypt** button on the Kleopatra toolbar and select the **sample.key** file. When the **Sign / Encrypt Files** window appears, specify the previously created key pair for **Sign as** and **Encrypt for me**.

- Sign as: Specify you own key pair.
- Encrypt for me: Specify you own key pair.
- Encrypt for others: Specify the Renesas PGP public key.

For **Output files/folder:** specify the output destination folder, then click the **Sign / Encrypt** button. The PGP encrypted UFPK file **sample.key.pgp** is created in the specified folder.

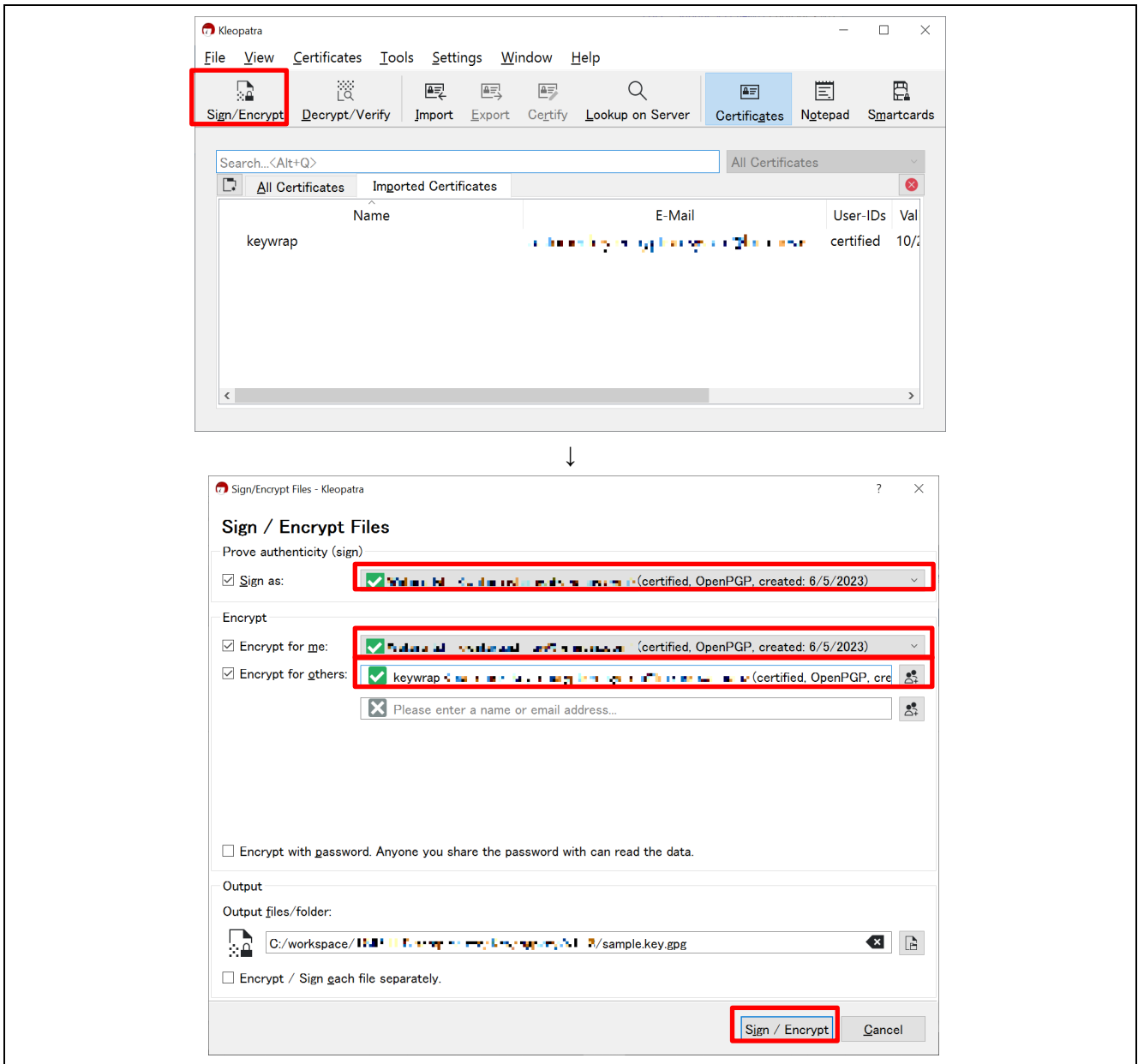


Figure 2-34 PGP Encrypting sample.key

## (6) Encrypting the PGP Encrypted UFPK on the DLM Server

Next, use the [Renesas Key Wrap service website](#) to upload the PGP encrypted UFPK to the DLM server and encrypt it. On the Key Wrap service website, click the following links in the order shown: **RENESAS RX** → **RX65N/RX651 Encryption of customer's data**\*1 → **Encryption service for products**. When the upload page appears, click the **Reference** button, specify the **sample.key.pgp** file created as described in (4), and click the **Settle** button.

Note: 1. If you are using the RX72N Envision Kit, click **RX72N Encryption of customer's data**.

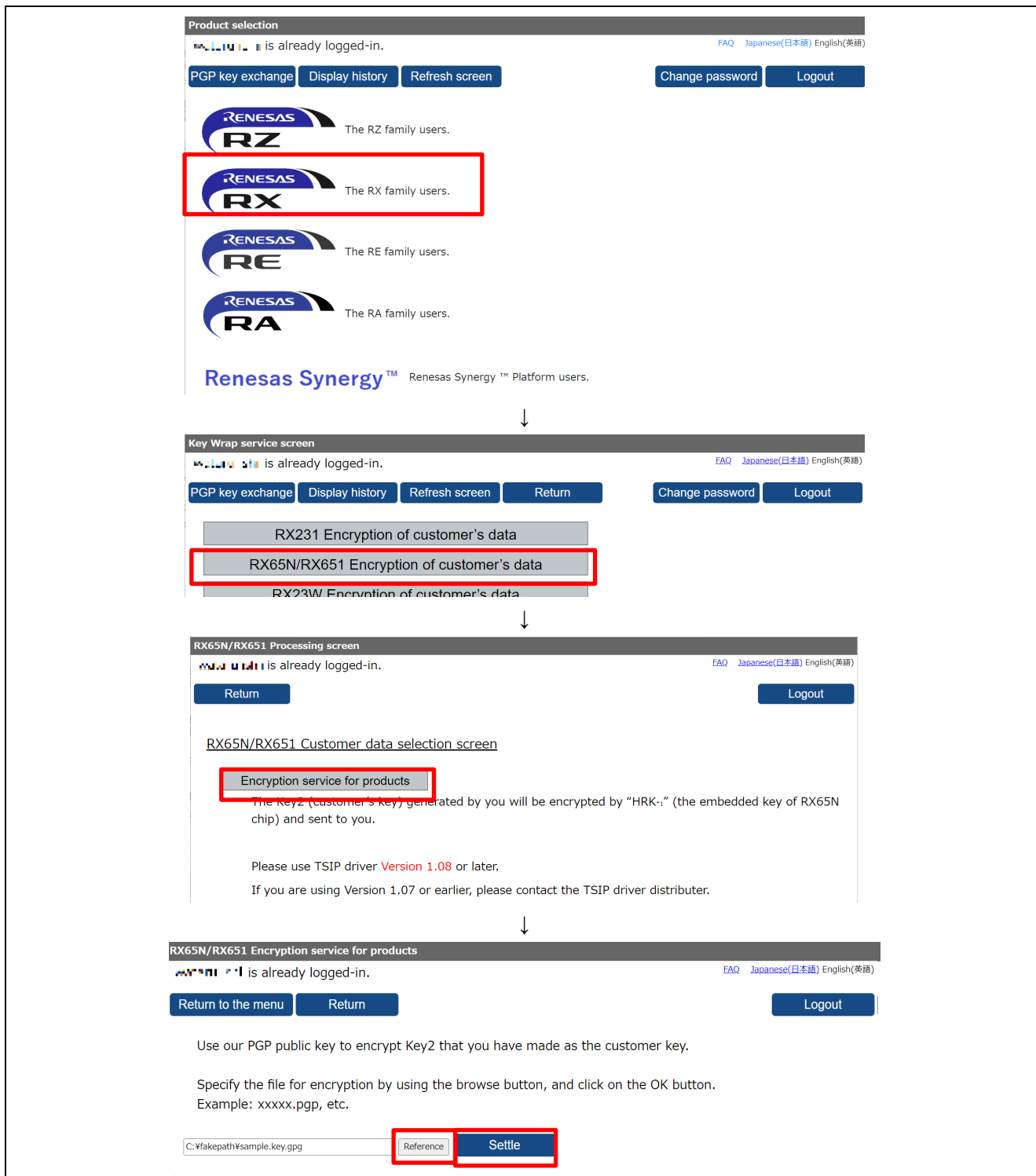


Figure 2-35 Uploading the UFPK



## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

The following page is displayed when the upload finishes, and encryption is performed on the Renesas DLM server. When encryption completes, a file named **sample.key\_enc.key.pgp** is emailed to you by Renesas. Store this file in a folder of your choice.

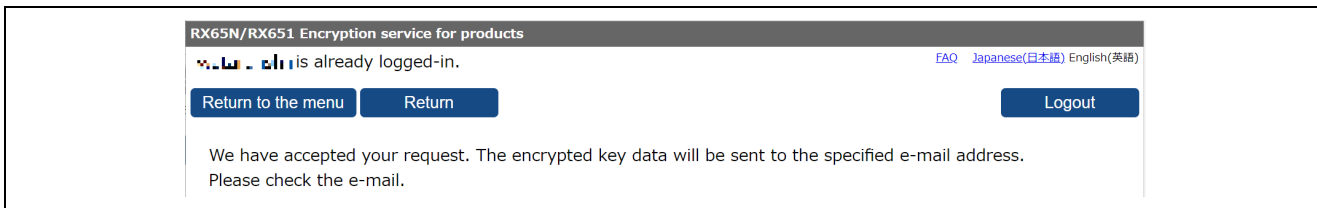


Figure 2-36 Upload to DLM Server Complete

### (7) Decrypting sample.key\_enc.key.pgp using OpenPGP

Use your own OpenPGP key to decrypt the file **sample.key\_enc.key.pgp** sent by Renesas to obtain the encrypted UFPK (W-UFPK). In Kleopatra, click the **Decrypt/Verify** button and select the file **sample.key\_enc.key.pgp** to commence decryption. When decryption finishes and the message **All operations completed.** appears in the window, click the **Save All** button to save the decrypted key file. The decrypted file is output to the same folder as that containing the encrypted file **sample.key\_enc.key**. This completes the process of encrypting **sample.key**.

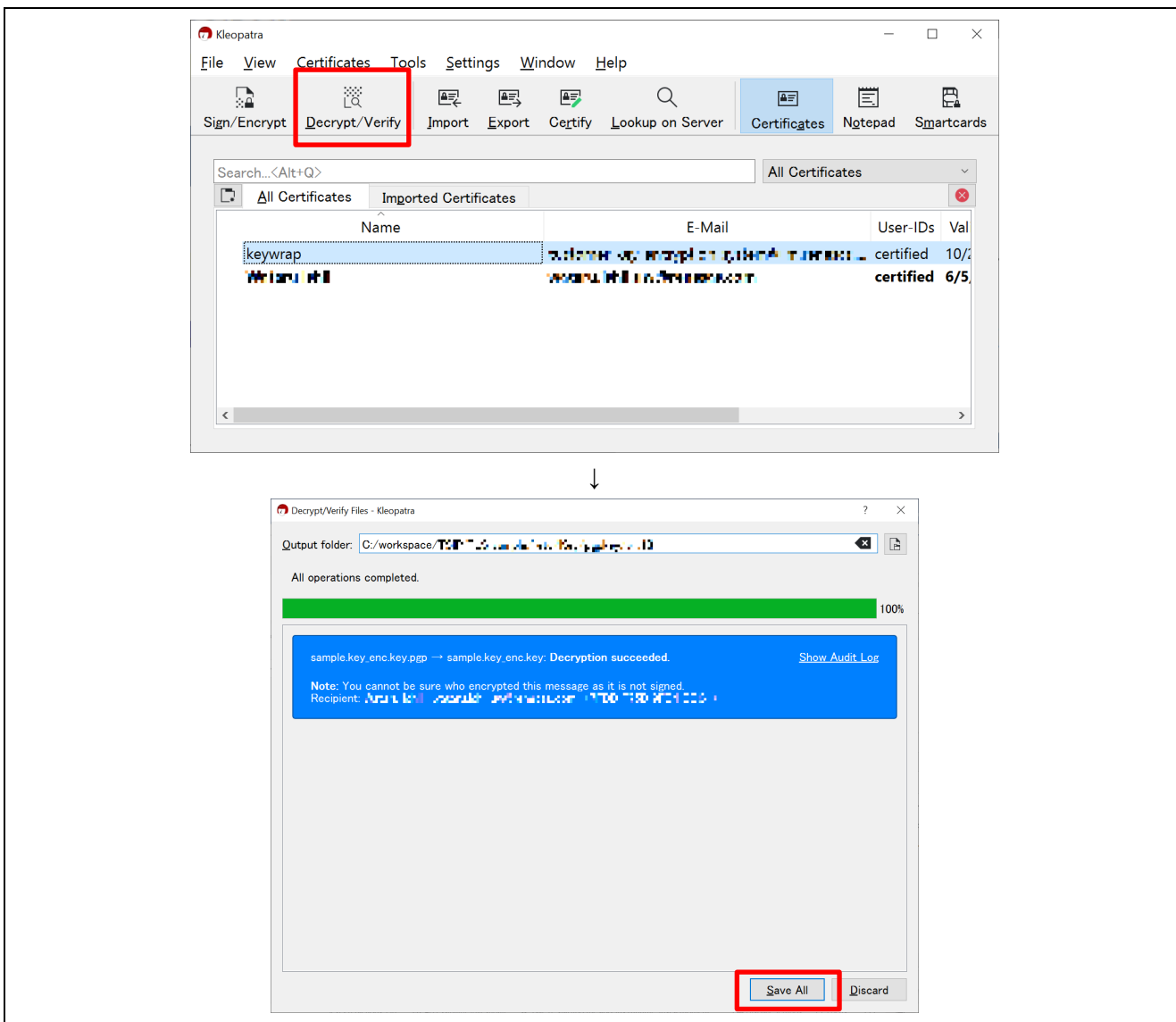


Figure 2-37 PGP Decryption of W-UFPK

### 2.4.5.2 Wrapping the Keying Data

Use the **sample.key** and **sample.key\_enc.key** files created as described in 2.4.5.1, Creating a UFPK and W-UFPK, to wrap the keying data for root CA certificate signature verification and the client certificate keying data and then convert the data for registration in the project files.

#### (1) Extracting Keying Data

Extract the keying data for root CA certificate signature verification and the client certificate keying data from the PEM format key files created as described in 2.4.3, Generating RSA Keys and Client Certificate, and 2.4.4, Root CA Certificate Signature Generation and Certificate File Format Conversion. Here we use the following two key files.

1. Public key file for root CA signature verification (PEM format)  
/key\_crt\_sig\_generator /ca-sign-keypair-rsa2048 /rsa2048-public.pem
2. Client certificate key pair file (PEM format)  
/key\_crt\_sig\_generator /client-rsa2048 /device1-private.pem.key

#### (2) Wrapping the Keying Data and Outputting Encrypted Key Files

Prepare the four data items listed below, created in (1) above and in 2.4.5.1. Then, enter these items in the Security Key Management Tool to generate encrypted key files. The resulting encrypted key files will be output as source code that can be incorporated into the project files as encrypted keying data wrapped using the UFPK and W-UFPK.

- UFPK (sample.key)
- W-UFPK (sample.key\_enc.key)
- Public key file for root CA certificate signature verification
- Client certificate key pair file

The following shows the procedure for generating encrypted key files.

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

1. Start the Security Key Management Tool that has already been installed.  
In the Security Key Management Tool app, open the **Wrap Key** tab. Then, open the **Key Type** tab, and then select the **RSA** radio button and **2048 bits, public**.

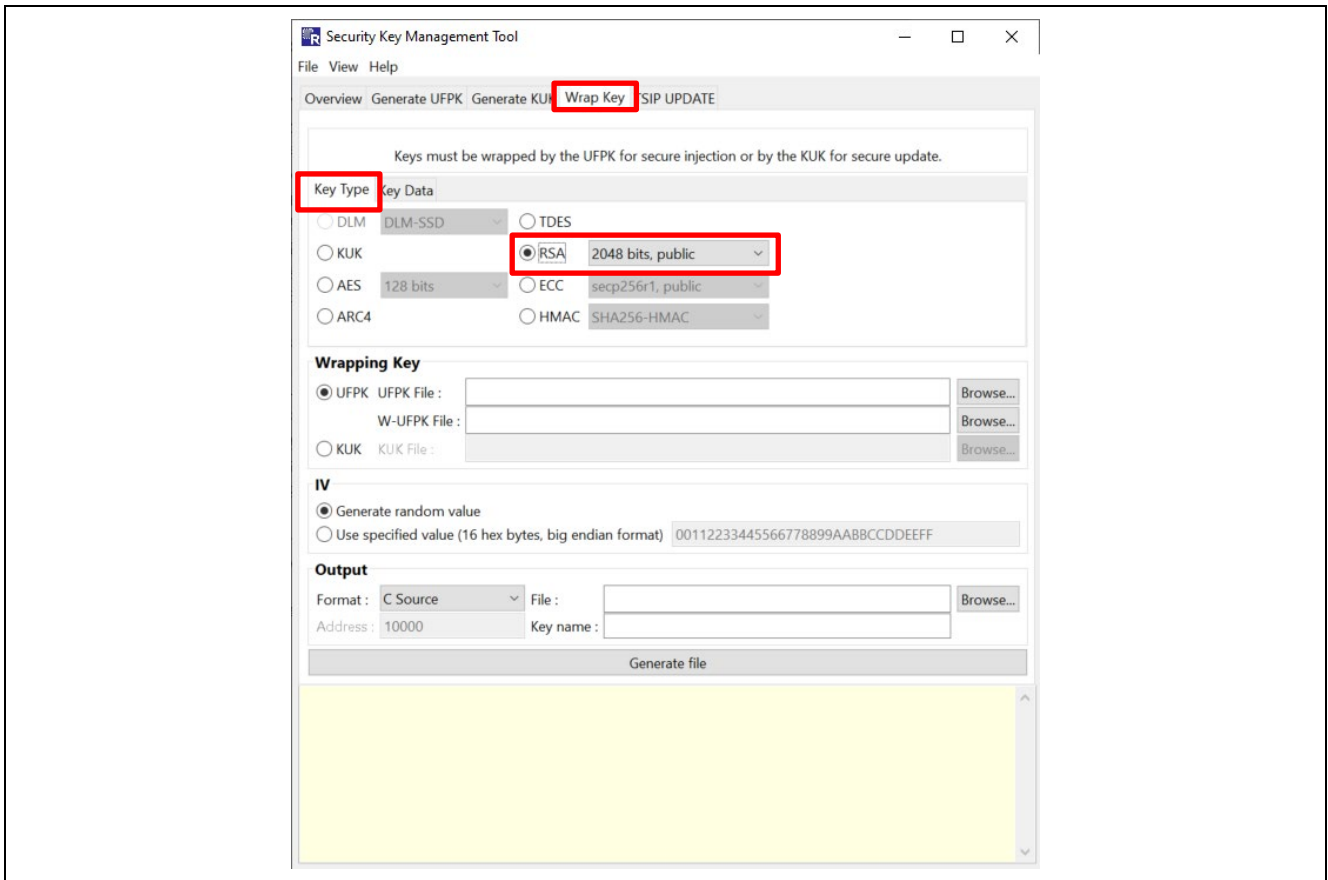


Figure 2-38 Wrapping Public Keys by Using the Security Key Management Tool

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

### 2. Register UFPK and W-UFPK files.

In the **Key Data** tab, in the **Wrapping Key** area, select the **UFPK** radio button. Then, click the **Browse** button for both the **UFPK File** and **W-UFPK File** fields, and then respectively specify the UFPK file (**sample.key**) and W-UFPK file (**sample.key\_enc.key**) that you created in 2.4.5.1.

In the **IV** area, select the **Generate random value** radio button. A random value is generated.

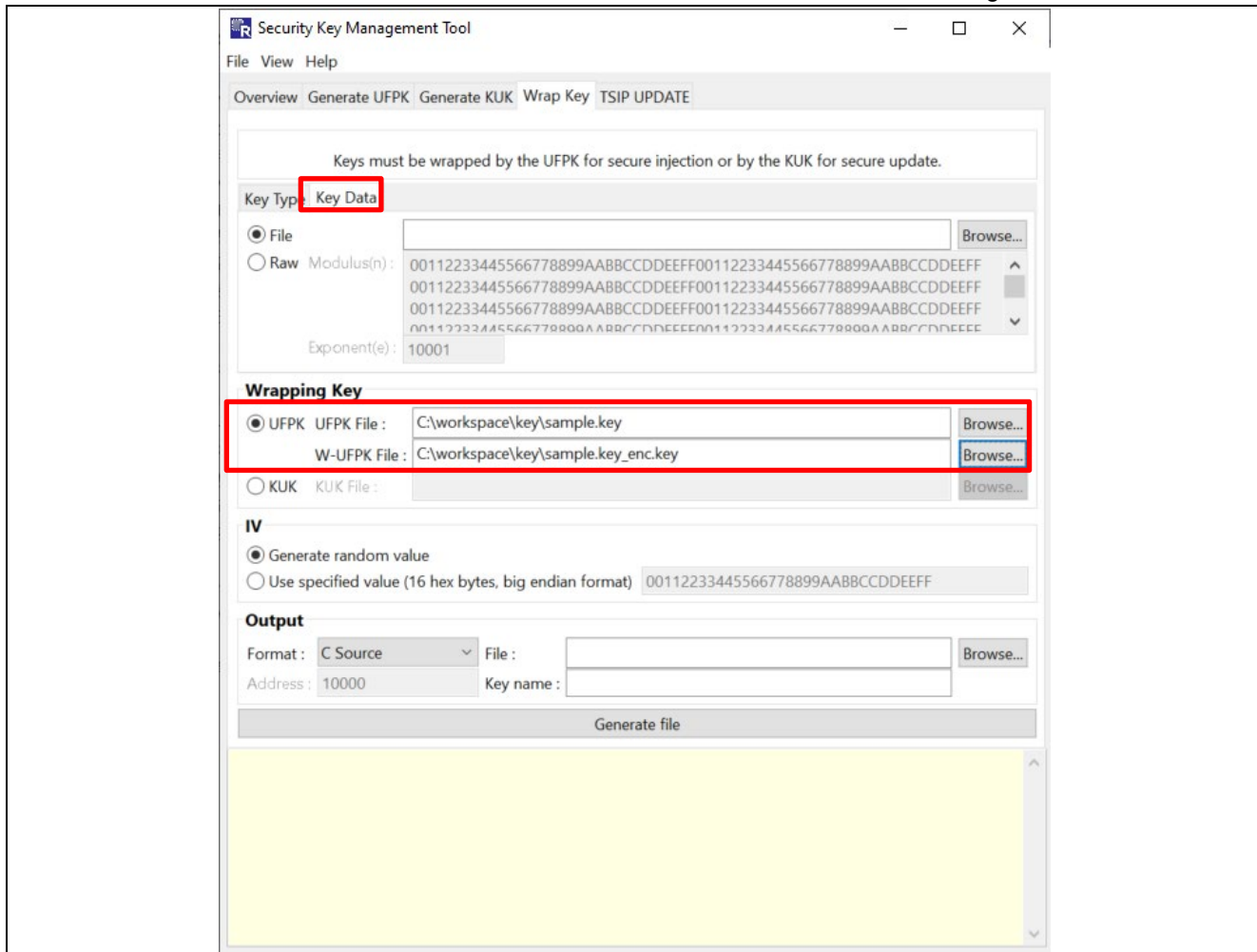


Figure 2-39 Specifying the UFPK and W-UFPK Files

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

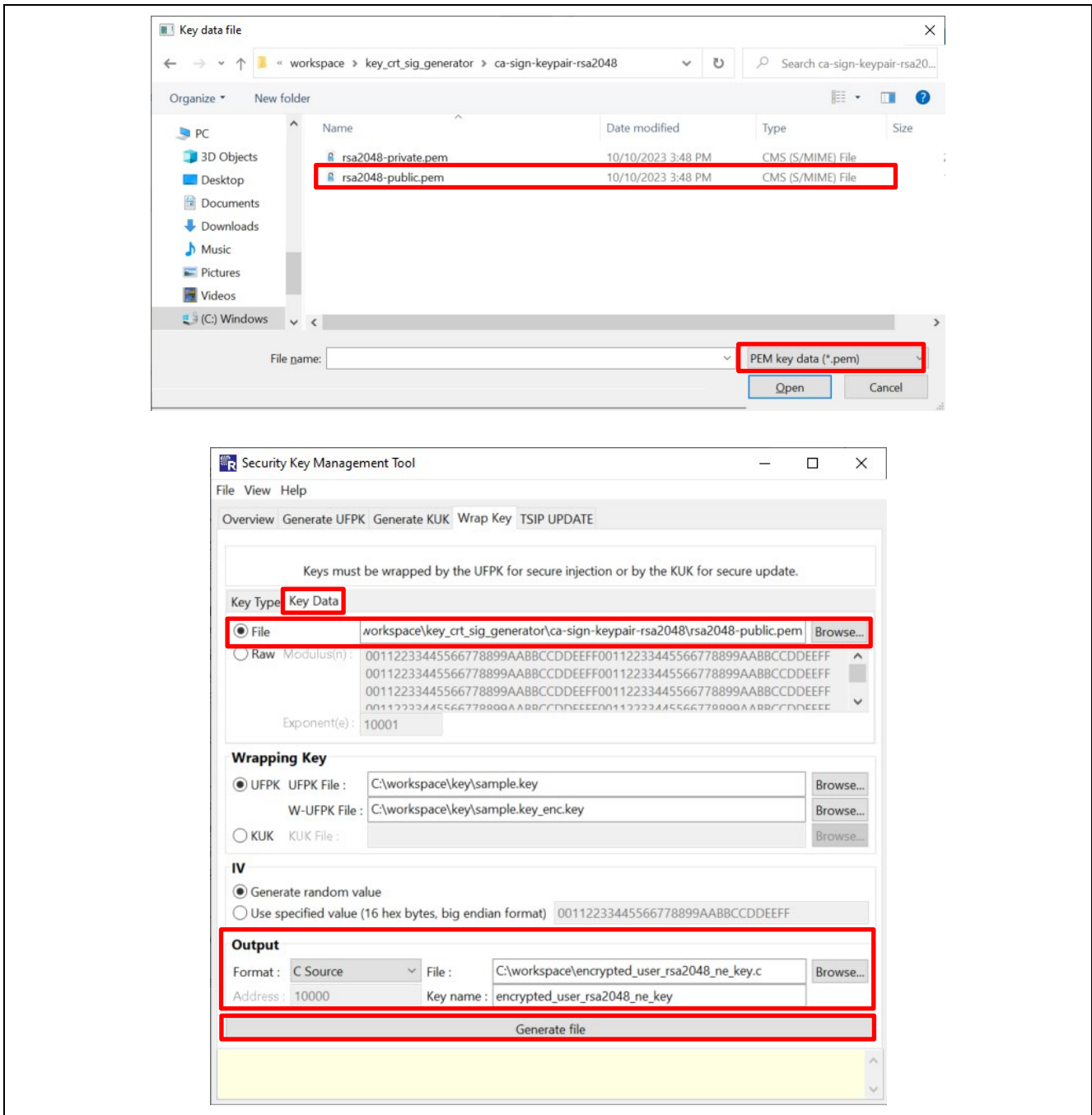
3. Generate the public key data for verifying the signature of the root CA certificate.

In the **Key Data** tab, select the **File** radio button. Then, click the **Browse** button, select **PEM key data (\*.pem)**, and then select the following file as the key pair file for root CA signature verification (in PEM format):

/key crt sig generator/ca-sign-keypair-rsa2048/rsa2048-public.pem

In the **Output** area, select a file named **encrypted\_user\_rsa2048\_ne\_key** in any folder of your choice in the **File** field. Similarly, in the **Key name** field, enter **encrypted\_user\_rsa2048\_ne\_key**. Then, click the **Generate file** button. The public key data for verifying the signature of the root CA certificate will be generated.

The string **encrypted\_user\_rsa2048\_ne\_key** (specified in the **File** and **Key name** fields) is hard-coded in the source code, and should therefore not be changed.



**Figure 2-40** Generating Public Key Data for Verifying the Signature of the Root CA Certificate

When the “**OPERATION SUCCESSFUL**” message appears at the bottom of the tool window, generation is complete.

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

### 4. Generate the public key for the client certificate.

Before performing this step, make sure that you rename the file in the `/key crt_sig_generator /client-rsa2048` folder as follows:

Before the change: `device1-private.pem.key`

After the change: `device1-private.pem`

Open the **Key Type** tab, and then make sure that the **RSA** radio button and **2048 bits, public** are selected in the same way as in step 1.

Open the **Key Data** tab again, and then select the **File** radio button, click the **Browse** button, select **PEM key data (\*.pem)**, and then select the `device1-private.pem` file, which is the client certificate key pair file (in PEM format) that you renamed.

In the **Output** area, select a file named `encrypted_user_rsa2048_ne_key2` in any folder of your choice in the **File** field. Similarly, in the **Key name** field, enter `encrypted_user_rsa2048_ne_key2`. Then, click the **Generate file** button. The public key data for the client certificate will be generated.

The string `encrypted_user_rsa2048_ne_key2` (specified in the **File** and **Key name** fields) is hard-coded in the source code, and therefore should not be changed.

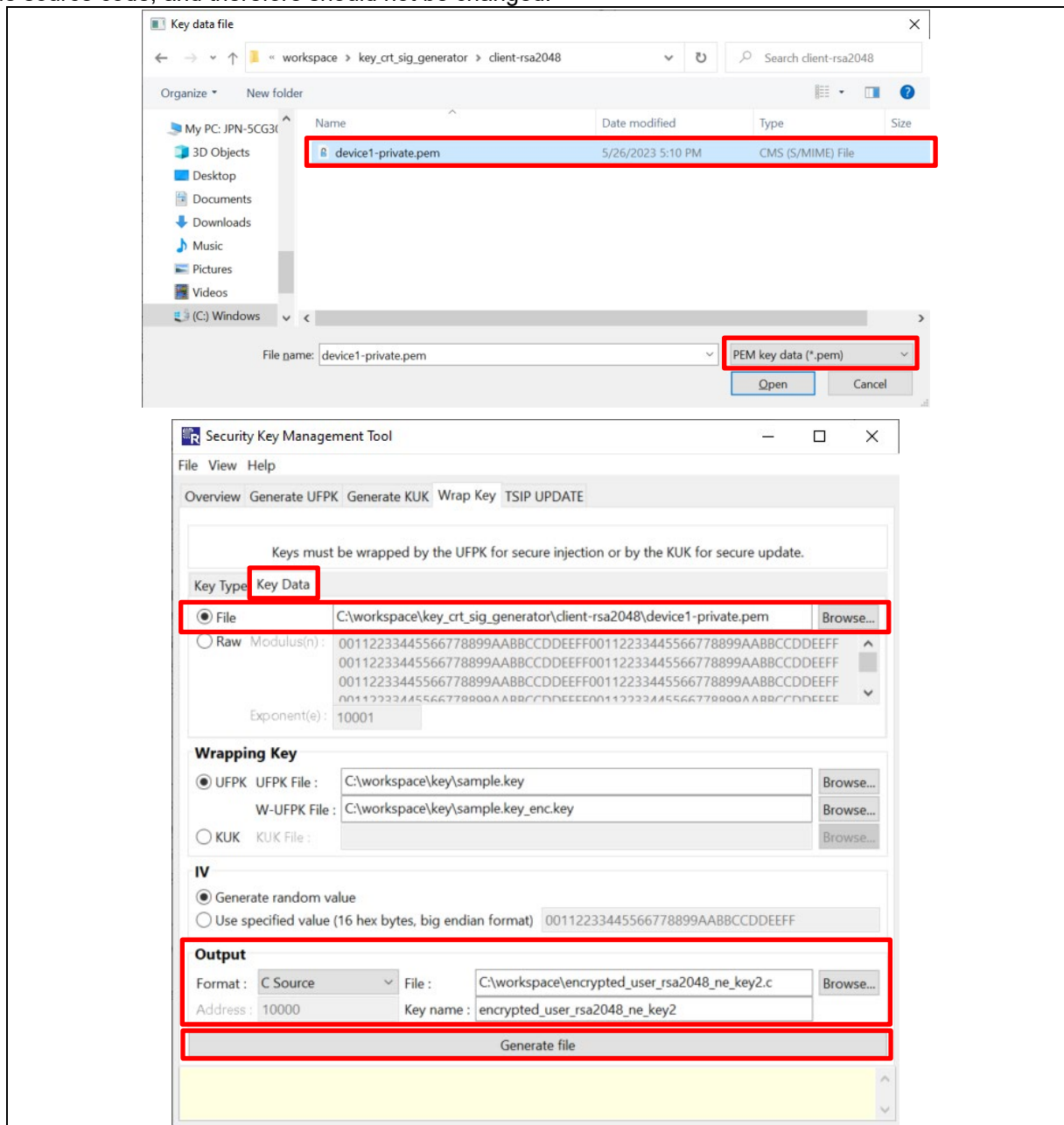
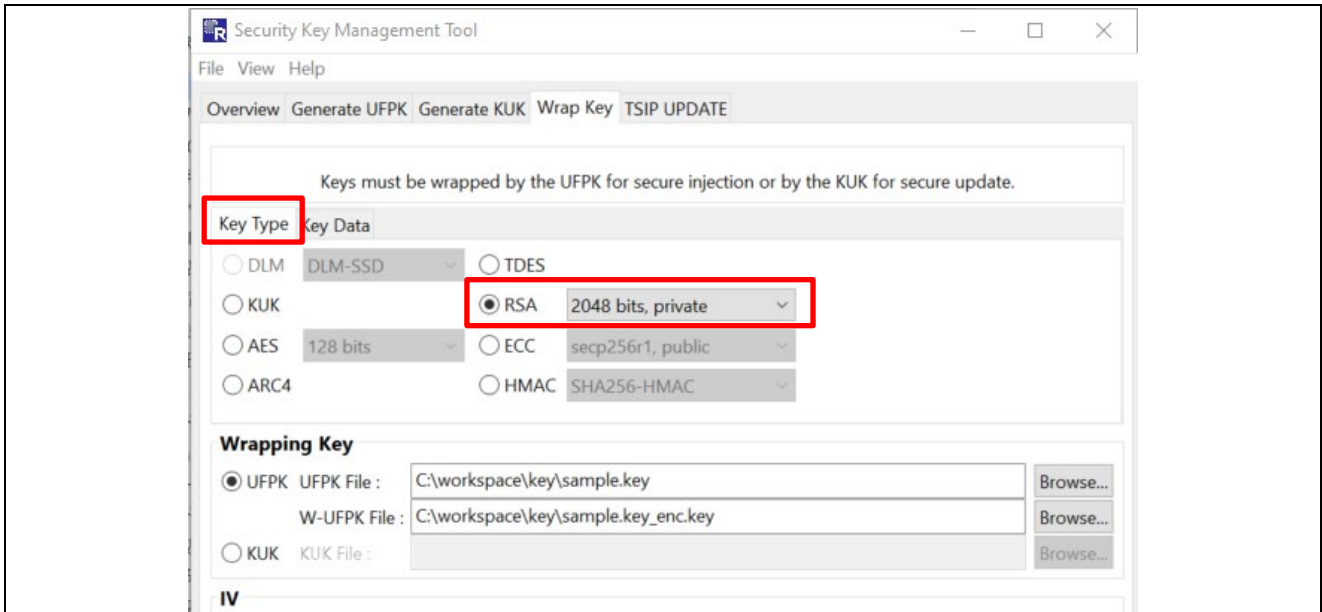


Figure 2-41 Generating the Public Key for the Client Certificate

When the “**OPERATION SUCCESSFUL**” message appears at the bottom of the tool window, generation is complete.

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

5. Generate the private key for the client certificate.  
Open the **Key Type** tab, and then select the **RSA** radio button and **2048 bits, private**.



**Figure 2-42 Private Key for the Client Certificate**

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

Open the **Key Data** tab again. Select the **File** radio button, click the **Browse** button, select **PEM key data (\*.pem)**, and then select the **/key crt\_sig\_generator /client-rsa2048/device1-private.pem** file, which is the client certificate key pair file (in PEM format) that you renamed.

In the **Output** area, select a file named **encrypted\_user\_rsa2048\_nd\_key** in any folder of your choice in the **File** field. Similarly, in the **Key name** field, enter **encrypted\_user\_rsa2048\_nd\_key**. Then, click the **Generate file** button. The private key data for the client certificate will be generated. The string **encrypted\_user\_rsa2048\_nd\_key** (specified in the **File** and **Key name** fields) is hard-coded in the source code, and therefore should not be changed.

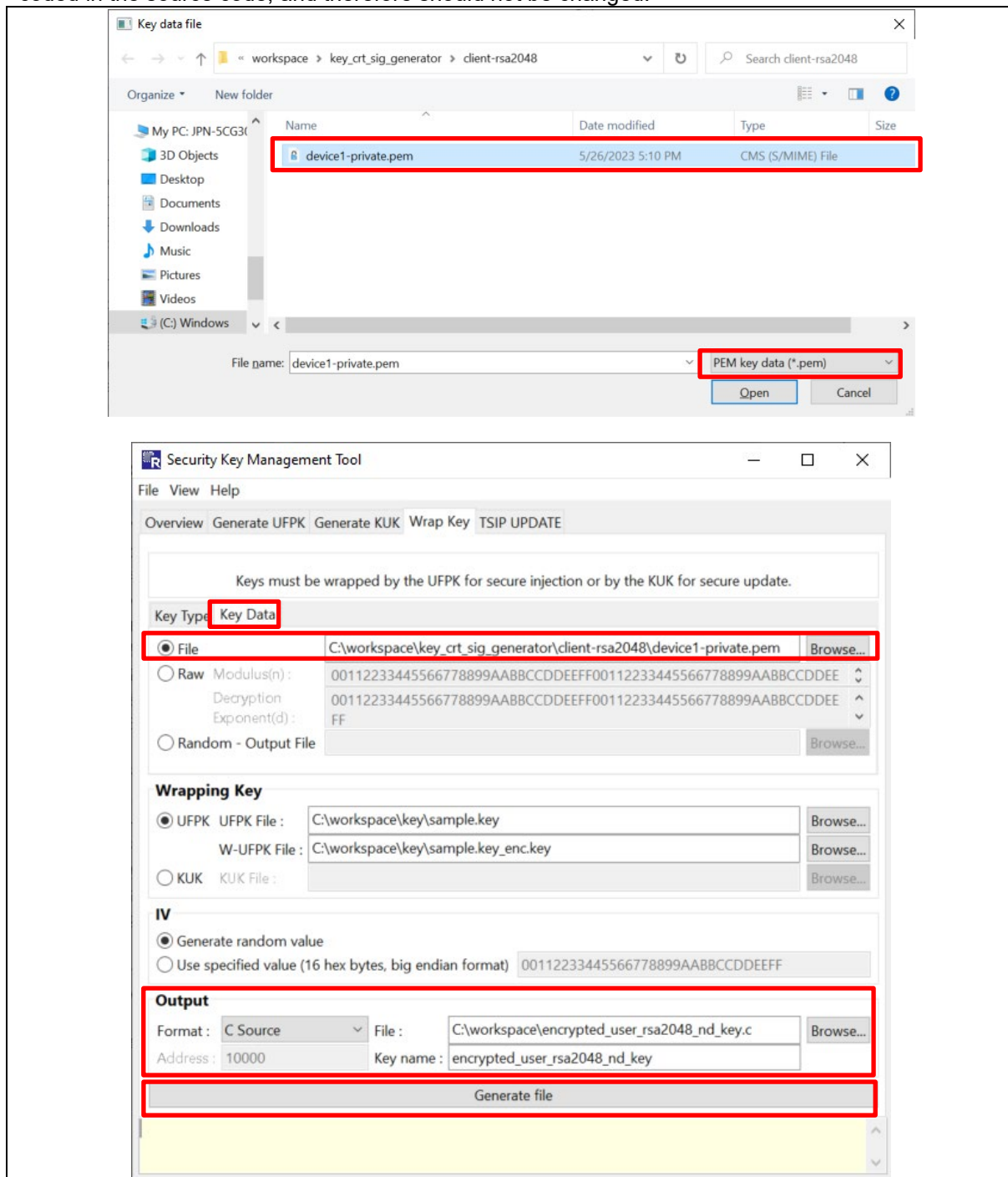


Figure 2-43 Wrapping Private Keys by Using the Security Key Management Tool

When the “**OPERATION SUCCESSFUL**” message appears at the bottom of the tool window, generation is complete.



6. Now you have the following six generated encrypted key files. Save these files in the **sample\_azure\_iot\_embedded\_sdk/src/userdata\_tsip** folder in the sample project. Overwrite the files existing in the folder with the new files.

- encrypted\_user\_rsa2048\_ne\_key.c
- encrypted\_user\_rsa2048\_ne\_key.h
- encrypted\_user\_rsa2048\_ne\_key2.c
- encrypted\_user\_rsa2048\_ne\_key2.h
- encrypted\_user\_rsa2048\_nd\_key.c
- encrypted\_user\_rsa2048\_nd\_key.h

This completes the procedure for preparing the basic project files. Section 3 describes how to configure settings on Microsoft Azure Portal and then register Azure-related setting values, and section 4 explains how to build and run the project.

## 3. Operations on Microsoft Azure Portal

The operations that you will need to perform on Microsoft Azure in order to run the sample project described in this document are explained below. The sample project uses IoT Hub Device Provisioning Service (DPS) to connect to an Azure IoT hub.

### 3.1 Preparations for Connection to Azure IoT Hub (Azure Portal)

#### 3.1.1 Creating an IoT Hub

Perform the preparations necessary for connection to Microsoft Azure. Follow the instruction in 3.1, Azure Preparation (3.1.1 and 3.1.2), in the application note [Visualization of Sensor Data using RX65N Cloud Kit and Azure RTOS](#) to create an IoT hub. The procedure described in this document connects to the IoT hub via DPS, so it is not necessary to create a device as described in 3.1.3, Create an IoT Device.

#### 3.1.2 Creating an IoT Hub Device Provisioning Service (DPS) Instance

Follow the instruction on the page linked to below to create a new IoT Hub Device Provisioning Service instance.

<https://learn.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision#create-a-new-iot-hub-device-provisioning-service-instance>

After creating the new DPS instance, follow the instructions on the page linked to below to link the IoT hub to the Device Provisioning Service instance.

<https://learn.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision#link-the-iot-hub-and-your-device-provisioning-service-instance>

This completes the procedure for creating an IoT hub and DPS instance.

#### 3.1.3 Device Provisioning Using the IoT Hub and DPS Instance

The steps for performing device provisioning using the newly created IoT hub and Device Provisioning Service (DPS) instance are described below.

1. On the Azure Portal home page, click **All services** → **Internet of Things** category → **Azure IoT Hub Device Provisioning Services**. On the list of **Azure IoT Hub Device Provisioning Services** that is displayed, select the DPS instance created as described in 3.1.2, Creating an IoT Hub Device Provisioning Service (DPS) Instance.

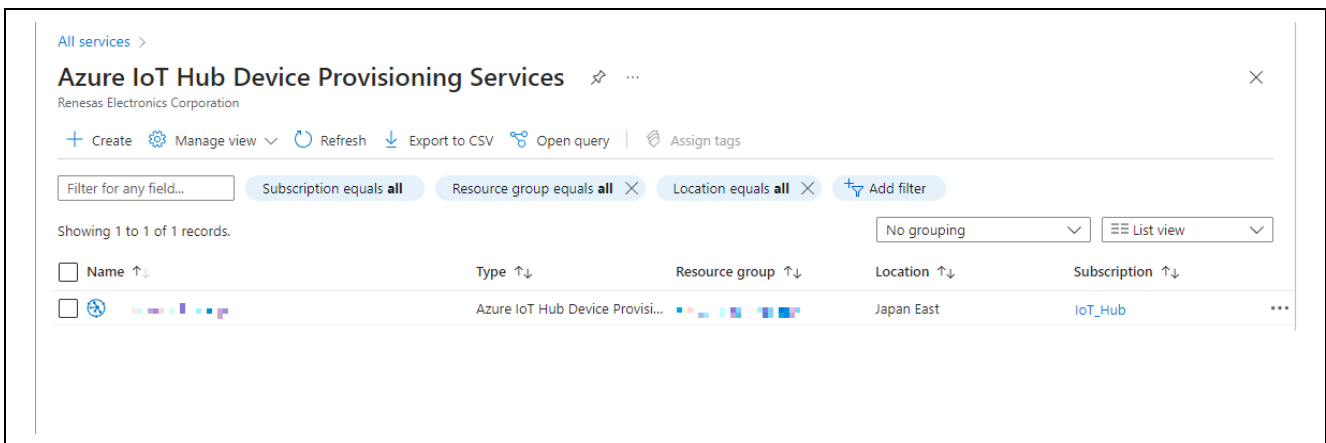


Figure 3-1 List of DPS Instances

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

2. On the menu panel of the selected DPS instance, click **Manage enrollments**. The **Manage enrollments** page is displayed.

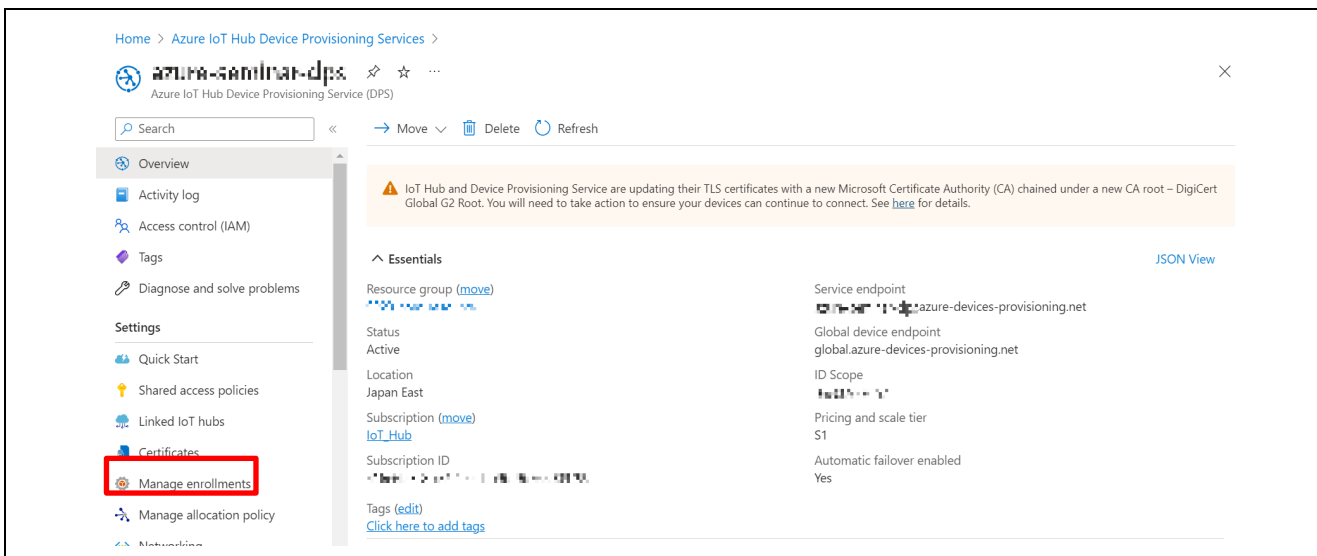


Figure 3-2 Managing DPS Enrollments

3. On the **Manage enrollments** page, click **Individual enrollments** tab → **Add individual enrollment** to display the **Add enrollment** page.

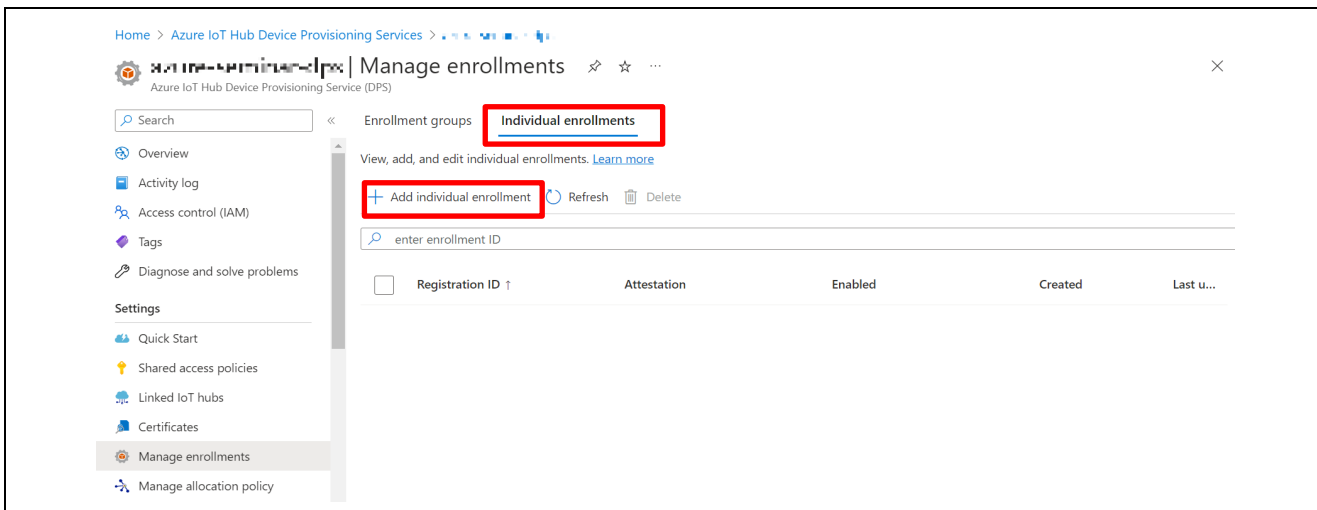



Figure 3-3 Manage enrollments Page

4. On the **Add enrollment** page, configure settings for the DPS instance. Enter the following items. When no value is specified, use the default setting.
  1. On the **Registration + provisioning** tab, configure the following settings.
    - For Attestation mechanism, specify X.509 client certificates.\*1
    - Click the folder icon  to the right of **Primary certificate file** and select the client certificate file. The client certificate file is **device1-certificate.pem.crt**, created as described in 2.4.3(1) 5. The file name extension must be PEM or CER, so first copy the client certificate file to a folder of your choice and rename it to **device1-certificate.pem** before selecting it. When the file has been selected, the **Common Name** specified in the certificate is displayed under **Primary certificate file**. After configuring settings, click the **Next: IoT hubs >** button.

# RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

Note: 1. If **Symmetric key** is selected, you will also need to specify an ID of your choice for **Registration ID** and specify a symmetric key in the project.

Home > Azure IoT Hub Device Provisioning Services > Manage enrollments >

## Add enrollment

Registration + provisioning | IoT hubs | Device settings | Review + create

### Attestation

Attestation is the process of verifying a device's identity during registration. Devices must attest their identity using the enrollment's selected attestation mechanism.

Attestation mechanism \*

X.509 client certificates

### X.509 certificate settings

Using client X.509 certificate attestation, Device Provisioning Service verifies a device's certificate against enrollment certificates. Enrollments may have one or two certificates. Uploaded certificates must share a common name.

Primary certificate file

device1-certificate.pem

Common name: TSIP\_sample\_test

Secondary certificate file

select certificate file

### Registration ID

Each registered device is assigned unique registration ID in Device Provisioning Service. The ID format will vary by attestation mechanism.

The registration ID will match the subject common name on selected certificates.

### Provisioning status

You can enable or disable this enrollment from provisioning and reprovisioning devices.

Enable this enrollment

### Reprovision policy

Provisioned devices may trigger requests to be reprovisioned. Reprovision policy specifies whether to reprovision the device and how handle the device's existing state data.

Reprovision policy

Reprovision device and migrate current state

Review + create | < Previous | Next: IoT hubs >

Figure 3-4 Add enrollment: Registration + provisioning

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

2. On the **IoT hubs** tab, enter the name of the IoT hub to be used for device provisioning under **Target IoT hubs**. After configuring settings, click the **Next: Device settings >** button.

Home > Azure IoT Hub Device Provisioning Services > Manage enrollments >

### Add enrollment

Registration + provisioning | **IoT hubs** | Device settings | Review + create

**Target IoT hubs**  
You can specify a set of linked IoT hubs where device(s) will be provisioned. If no IoT hubs are selected, devices may be provisioned in any linked IoT hub.

Target IoT hubs

Add link to IoT hub

**Allocation policy**  
Allocation policy determines which target IoT hub a device is assigned when provisioned. The default allocation policy is configured under 'Manage allocation policy.'

Allocation policy \*

Static

Evenly weighted distribution

Lowest latency

Custom (use Azure Function)

[Review + create](#) | [< Previous](#) | [Next: Device settings >](#)

**Figure 3-5 Add enrollment: IoT hubs**

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

3. On the **Device settings** tab, specify the **Device ID**.

It is not necessary to enter a setting for device ID because the **Common Name** registered in the primary certificate file as described in 2.4.3(1) is already set as the device ID. If you wish to specify a different device ID, enter it under **Device ID**. After configuring settings, click the **Next: Review + create >** button.

Home > Azure IoT Hub Device Provisioning Services > Manage enrollments >

### Add enrollment

Registration + provisioning | IoT hubs | **Device settings** | Review + create

**Device ID**  
You can optionally specify the name assigned to a provisioned device. If no value is specified, the device ID will match the enrollment registration ID.

Device ID  
TSIP\_sample\_test

**IoT Edge**  
IoT Edge enabled devices can run Docker-compatible modules to perform additional work.  
 Enable IoT Edge on provisioned device

**Device tags**  
Device tags annotate devices with queryable properties and can be used to find or manage devices. You can specify tags to assign when a device is provisioned. Tags must be valid JSON.

Device tags \*

```
1 {}
```

**Review + create** | < Previous | **Next: Review + create >**

Figure 3-6 Add enrollment: Device settings

4. On the **Review + create** tab, check the registered settings and, if there are no problems, click the **Create** button.

Home > Azure IoT Hub Device Provisioning Services > Manage enrollments >

### Add enrollment

Registration + provisioning | IoT hubs | Device settings | **Review + create**

**Registration + provisioning**  
Attestation mechanism: X.509 client certificates  
Registration ID: TSIP\_sample\_test  
Enrollment enabled: Yes  
Reprovision policy: Reprovision device and migrate current state

**IoT hubs**  
Allocation policy: Evenly weighted distribution  
Target IoT hubs: azure-devices.net

**Device settings**  
Device ID: TSIP\_sample\_test  
IoT Edge: Not enabled

Initial twin

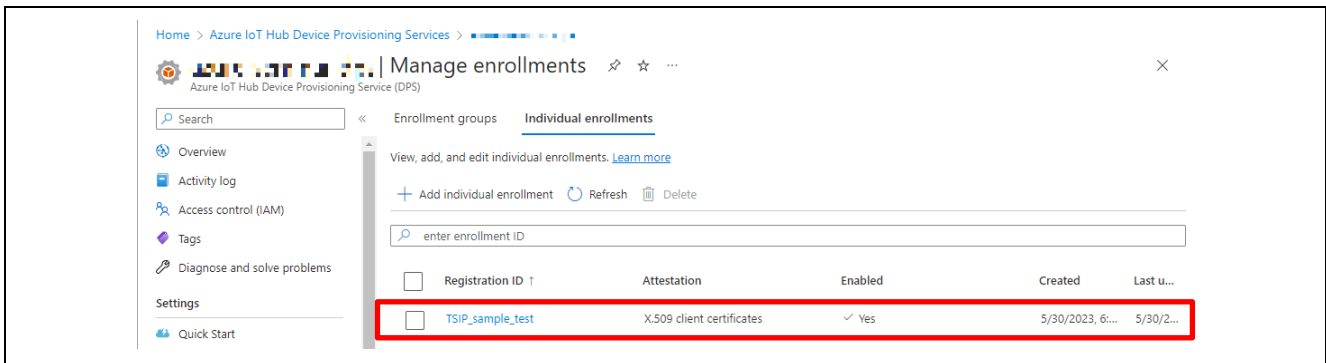
```
1 {  
2   "tags": {},  
3   "properties": {
```

**Create** | < Previous | Next >

Figure 3-7 Add enrollment: Review + create

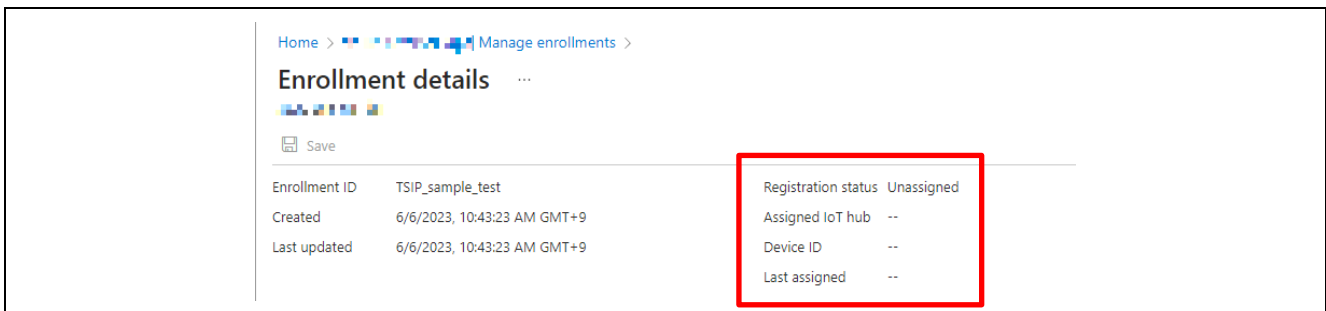
## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

- When the settings have been successfully saved, the newly created DPS instance is added to the list under **Registration ID** on the **Individual enrollments** tab.



**Figure 3-8 DPS Registration ID List**

This registration ID is the **Common Name** registered on the client certificate used. (If a user-defined device name was registered, the registered device name is used.) Make a note of the registration ID for later use because you will need to specify it in the project source code. In addition, clicking the registration ID displays **Enrollment details**, as shown below. The **Enrollment details** page is empty immediately after enrollment, but the details will be displayed once the DPS instance begins to function and completes enrollment on the IoT hub.



**Figure 3-9 Enrollment details**

This completes setting configuration on the Azure Portal.

### 3.2 Microsoft Azure Communication Settings

Configure settings in the sample project source code prepared as described in section 2.

#### 3.2.1 Azure IoT Settings

Open the file `src/sample_config.h` in the `sample_azure_iot_embedded_sdk` folder of the sample project, and configure the following settings.

##### (1) Parameter Settings

Define operating parameters for encryption, etc., in `sample_config.h` as shown below.

**Table 3-1 Parameter Settings**

Parameter	Setting Value	Setting Value in Sample	Remarks
SEL_CIPHER_SUITE	0: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (default) 1: Select cipher suite	0	
SEL_DEVICE_AUTH	0: Symmetric key 1: X.509 self-signed certificate 2: X.509CA signed certificate (not supported)	1	
SEL_DPS	0: IoT hub connection (no DPS connection) 1: DPS connection + IoT hub connection	1	
SEL_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA	0: Not specified by client Hello message 1: Specified by client Hello message	0	Setting has no effect when SEL_CIPHER_SUITE = 0.
SEL_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA	0: Not specified by client Hello message 1: Specified by client Hello message	0	
SEL_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256	0: Not specified by client Hello message 1: Specified by client Hello message	0	
SEL_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256	0: Not specified by client Hello message 1: Specified by client Hello message	0	
SEL_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	0: Not specified by client Hello message 1: Specified by client Hello message	0	

```

11  3      Copyright (c) Microsoft Corporation. All rights reserved.
12  11
13  12  #ifndef SAMPLE_CONFIG_H
14  13  #define SAMPLE_CONFIG_H
15  14
16  15  #ifndef __cplusplus
17  16  extern "C" {
18  17  #endif
19  18
20  19  #include "platform.h"
21  20  #if (BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED == 1)
22  21  #define SEL_CIPHER_SUITE 0 // 0:TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (default), 1:Select cipher suite
23  22  #define SEL_DEVICE_AUTH 1 // 0:Symmetric, 1:X.509 Self Signed Certificate, 2:X.509 CA Certificate
24  23  #define SEL_DPS 1 // 0:DPS Not Connect, 1:DPS Connect
25  24
26  25  #endif // SEL_CIPHER_SUITE
27  26  #define SEL_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA 0 // TLS_RSA_WITH_AES_128_CBC_SHA (No.1)
28  27  #define SEL_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA 0 // TLS_RSA_WITH_AES_256_CBC_SHA (No.2)
29  28  #define SEL_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 0 // TLS_RSA_WITH_AES_128_CBC_SHA256 (No.3)
30  29  #define SEL_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 0 // TLS_RSA_WITH_AES_256_CBC_SHA256 (No.4)
31  30  #define SEL_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 0 // TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (No.6)
32  31  #endif /* SEL_CIPHER_SUITE */
33  32
34  33  #endif /* BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED */
    
```

**Figure 3-10 Parameter Settings in sample\_config.h**



# RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

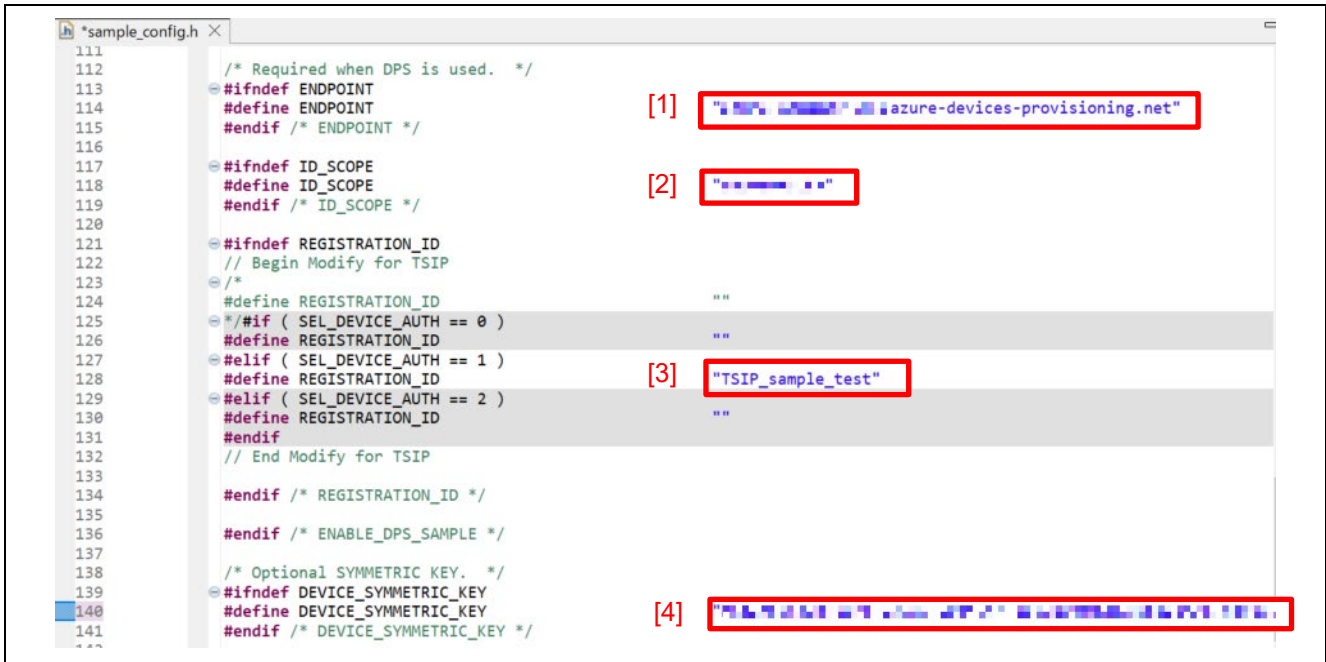
## (2) Azure IoT Hub DPS Authentication Information

Define in **sample\_config.h** the Azure IoT hub DPS enrollment information created as described in 3.1.

- [1] #define ENDPOINT: Service endpoint of connection target DPS
- [2] #define ID\_SCOPE: ID scope of connection target DPS
- [3] #define REGISTRATION\_ID: Registration ID of connection target DPS
- [4] #define DEVICE\_SYMMETRIC\_KEY: Primary key of connection target DPS\*1

Note: 1. The primary key only needs to be defined when **Symmetric key** was selected for **Attestation mechanism** during DPS enrollment.

Enter values in the above macros between quote marks (" "), as shown below.



```
111
112 /* Required when DPS is used. */
113 #ifndef ENDPOINT
114 #define ENDPOINT
115 #endif /* ENDPOINT */
116
117 #ifndef ID_SCOPE
118 #define ID_SCOPE
119 #endif /* ID_SCOPE */
120
121 #ifndef REGISTRATION_ID
122 // Begin Modify for TSIP
123 /*
124 #define REGISTRATION_ID
125 */
126 #if ( SEL_DEVICE_AUTH == 0 )
127 #define REGISTRATION_ID
128 #elif ( SEL_DEVICE_AUTH == 1 )
129 #define REGISTRATION_ID "TSIP_sample_test"
130 #elif ( SEL_DEVICE_AUTH == 2 )
131 #define REGISTRATION_ID
132 // End Modify for TSIP
133 #endif /* REGISTRATION_ID */
134 #endif /* ENABLE_DPS_SAMPLE */
135
136 /* Optional SYMMETRIC KEY. */
137 #ifndef DEVICE_SYMMETRIC_KEY
138 #define DEVICE_SYMMETRIC_KEY
139 #endif /* DEVICE_SYMMETRIC_KEY */
140
141
```

Figure 3-11 Authentication Information Settings in sample\_config.h

Confirm the above authentication information on the Azure Portal. On the Azure home page, select the DPS instance created as described in 3.1.2, then copy of setting values from the various pages shown and paste them into the source code. [1] **Service endpoint** and [2] **ID Scope** can be displayed by selecting the DPS instance and clicking **Overview** on the menu panel.

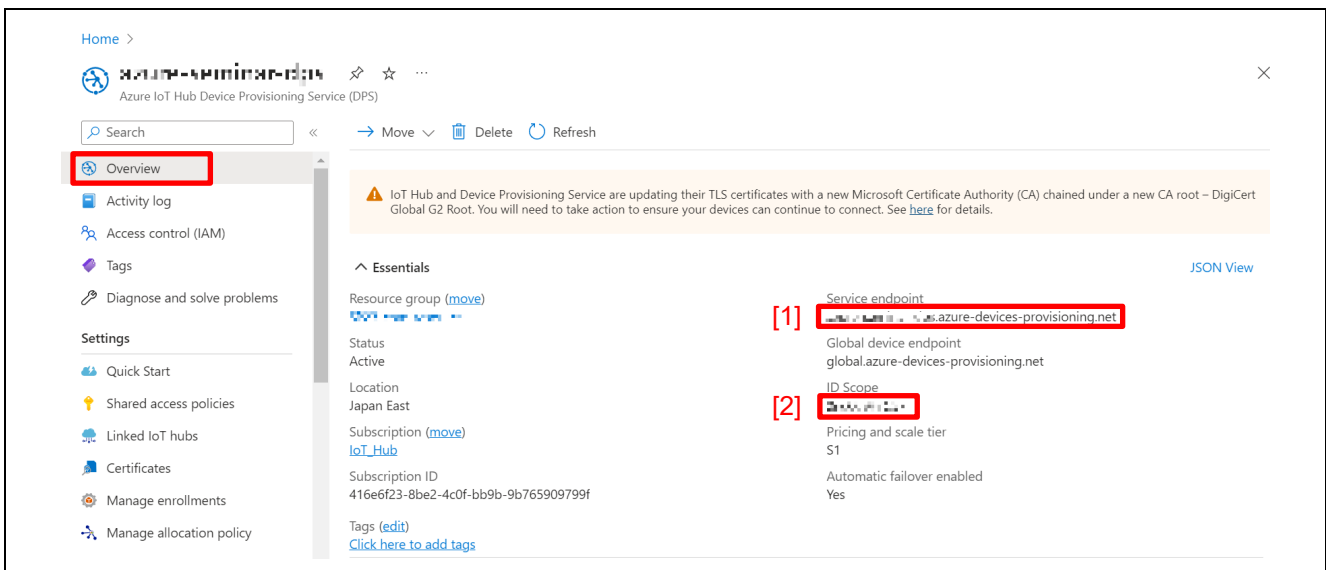
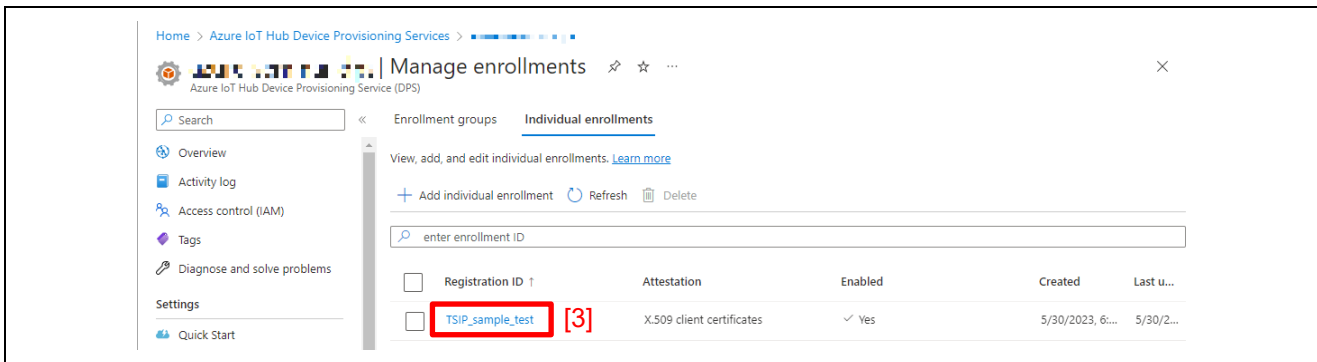


Figure 3-12 DPS Overview Page

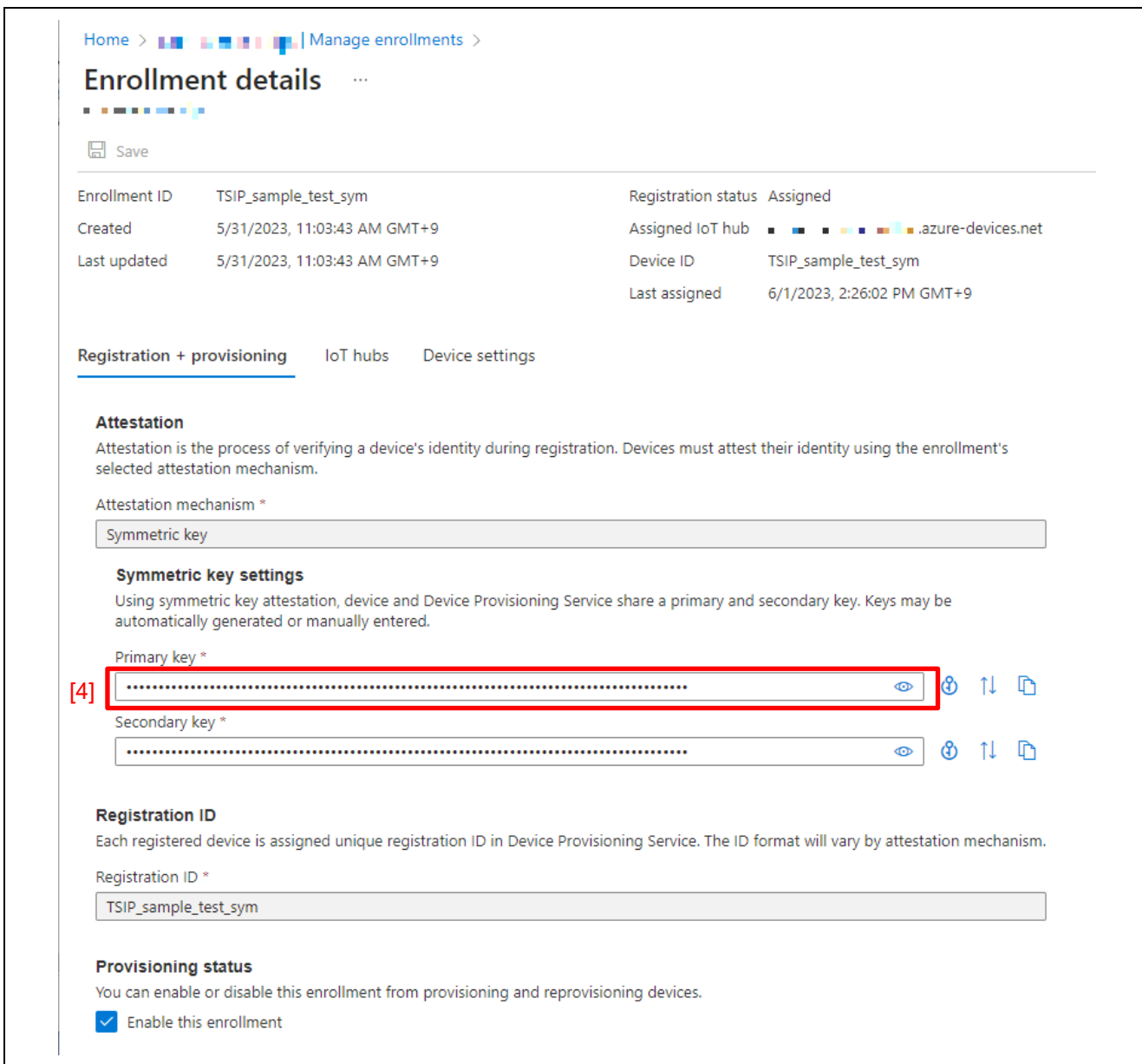
# RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

[3] **Registration ID** can be displayed by referring to the **Registration ID** list after selecting the **DPS** instance.



**Figure 3-13 DPS Registration ID List**

[4] **Primary key** only needs to be defined when **Symmetric key** was selected for **Attestation mechanism** (**SEL\_DEVICE\_AUTH** defined as **0**). Copy the **Primary key** from the **Enrollment details** page for a DPS registration ID with **Attestation mechanism** set to **Symmetric key**. This setting is not necessary when **Attestation mechanism** is set to **X.509 client certificates**.



**Figure 3-14 Enrollment details Page when Attestation Mechanism is Symmetric Key**

## (3) Connection Information When Not Using DPS

If no DPS connection is used, you will need to register a device on the IoT hub. Follow the steps in 3.1.3, Create an IoT Device, in [Visualization of Sensor Data using RX65N Cloud Kit and Azure RTOS](#) to register a device manually. In addition, you will need to configure IoT hub settings as described in step 8 of 3.2, Software Preparation, in the above document. Configure the following IoT hub setting in the sample project.

- HOST\_NAME
- DEVICE\_ID
- DEVICE\_SYMMETRIC\_KEY (when using a symmetric key)

Also, refer to Table 3.1, Parameter Settings, and configure the **SEL\_DPS** parameter for **no DPS connection**.

```

3      /* Copyright (c) Microsoft Corporation. All rights reserved. */
11
12     #ifndef SAMPLE_CONFIG_H
13     #define SAMPLE_CONFIG_H
14
15     #ifdef __cplusplus
16     extern "C" {
17     #endif
18
19
20     #include "platform.h"
21     #if ( BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED == true )
22     #define SEL_CIPHER_SUITE 0 // 0:TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (default),
23     #define SEL_DEVICE_AUTH 1 // 0:Symmetric, 1:X.509 Self Signed Certificate, 2:X.
24     #define SEL_DPS 0 // 0:DPS Not Connect, 1:DPS Connect
25
26
27     /* Required when DPS is not used. */
28     /* These values can be picked from device connection string which is of format : HostName=<host1>;Device
29     HOST_NAME can be set to <host1>,
30     DEVICE_ID can be set to <device1>,
31     DEVICE_SYMMETRIC_KEY can be set to <key1>. */
32     #ifndef HOST_NAME
33     #define HOST_NAME "azure-devices.net"
34     #endif /* HOST_NAME */
35
36     #ifndef DEVICE_ID
37     // Begin Modify for TSIP
38     /*
39     #define DEVICE_ID ""
40     */
41     #if ( SEL_DEVICE_AUTH == 0 )
42     #define DEVICE_ID "TSIP_sample_test_sym"
43     #elif ( SEL_DEVICE_AUTH == 1 )
44     #define DEVICE_ID "TSIP_sample_test"
45     #elif ( SEL_DEVICE_AUTH == 2 )
46     #define DEVICE_ID ""
47     #endif
48     // End Modify for TSIP
49     #endif /* DEVICE_ID */
50
51

```

Figure 3-15 Settings when Not Using DPS

## 3.2.2 IP Address Settings

When Ethernet-based communication is performed with the default settings of this sample project, DHCP is used for network connection. If the DHCP function is disabled on the router to which the target board is connected, configure settings as follows.

Open the file **src/main.c** in the **sample\_azure\_iot\_embedded\_sdk** folder, and add the line **#define SAMPLE\_DHCP\_DISABLE**.

Open the file **src/main.c** in the **sample\_azure\_iot\_embedded\_sdk** folder, and enter values for IP address, default gateway, DNS server address, and subnet mask.

If you use the CK-RX65N+RYZ014A (Cellular) board, you must specify the settings according to the SIM card you use. The procedure for specifying these settings is described later.

## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

1. In the **sample\_azure\_iot\_embedded\_sdk** folder, activate **sample\_azure\_iot\_embedded\_sdk.scfg**, and then open the **Components** tab.
2. In the tree view on the left of the tab, under the **RTOS Library** folder, click to open the **ewf** node.
3. On the right of the tab, for the property named **The SIM operator APN**, set the access point of the SIM card you use.
4. Save the file (**sample\_azure\_iot\_embedded\_sdk.scfg**), and then click the **Generate Code** button at the top right of the window to generate code.

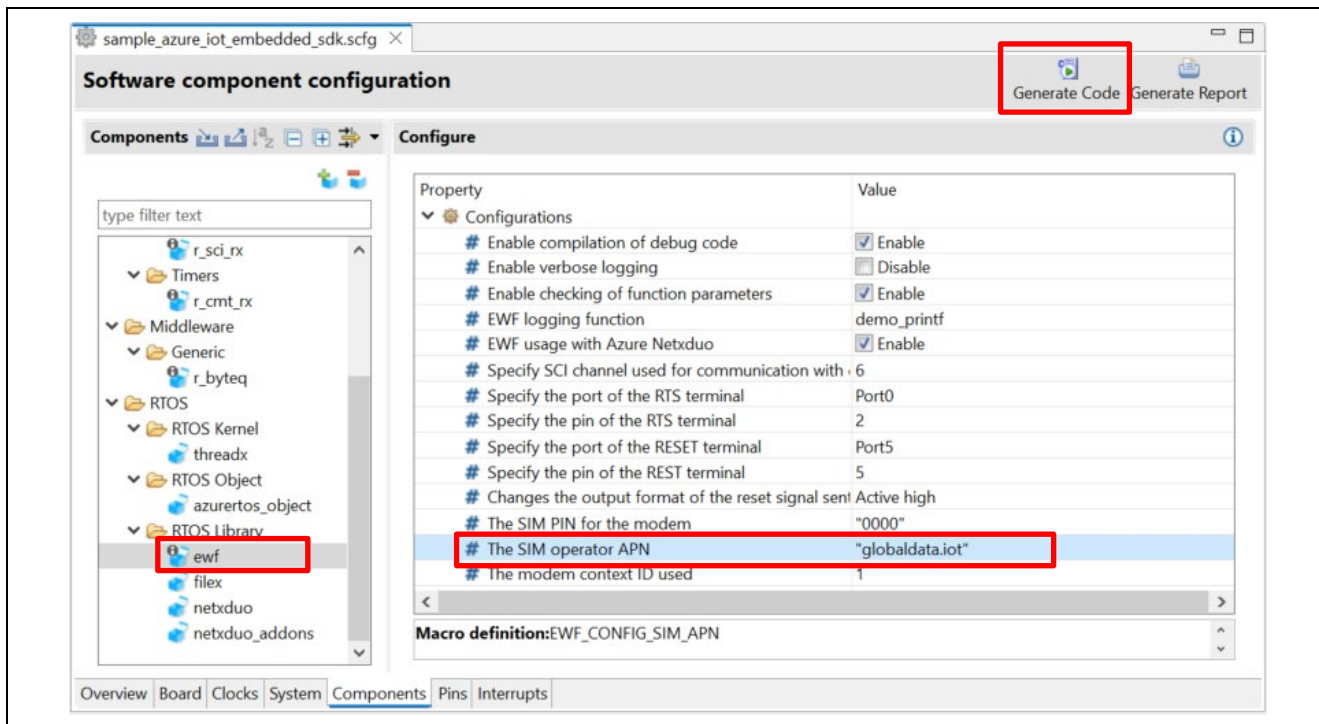


Figure 3-16 Setting the Access Point

### 3.2.3 Client Certificate Format Selection

By default, the RSA certificate type is enabled. ECDSA certificates are not supported by the sample project described in this document. The relevant definition is located in **src/userdata\_tsip/r\_trust\_certificate\_data.h** in the sample project folder.

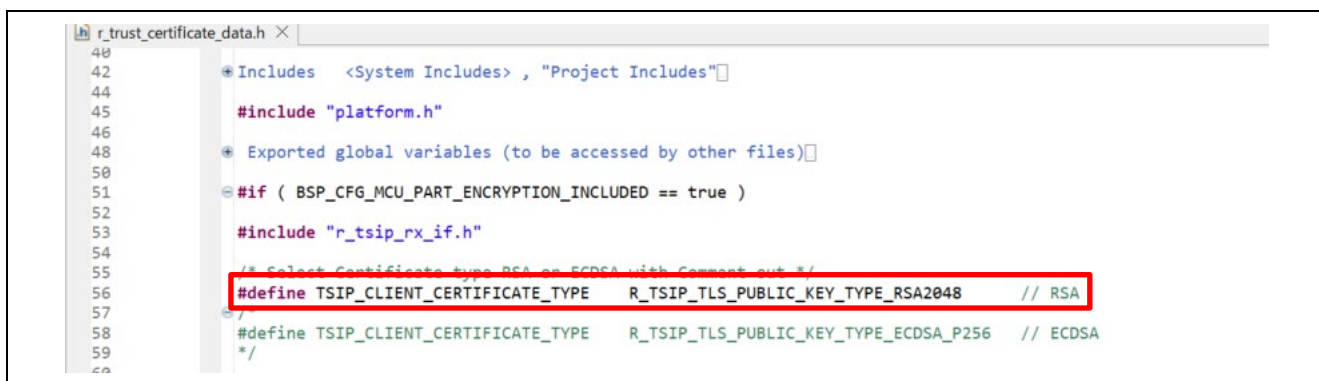


Figure 3-17 Client Certificate Format Selection

## 4. Building and Running the Project

Build the project created as described in sections 2 and 3. Before building the project, make sure to confirm the items described in section 4.1.

### 4.1 Items to Confirm Before Building the Project

#### 4.1.1 Settings for Renesas Starter Kit+ for RX65N-2MB

In the project of Renesas Starter Kit+ for RX65N-2MB, make sure that **true** is set for the following macro in the **sample\_azure\_iot\_embedded\_sdk/src/smc\_gen/r\_config/r\_bsp\_config.h** file. This macro is used for decision of whether to perform TSIP processing. If **false** is set, manually change it to **true**.

```
#define BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED
```

Note that the above setting is reset to **false** when you generate new code.

#### 4.1.2 Setting for Code Generation During Build

In this sample project, as described above, changes may have been made to some settings in files generated by Smart Configurator, depending on the board. Such additional settings may be lost when you generate new code. Therefore, perform steps 1 to 3 below to prevent all the source files from being regenerated when building or cleaning the project.

1. In e<sup>2</sup> studio's **Project Explorer** view, right-click the target project and select **Properties** from the context menu.

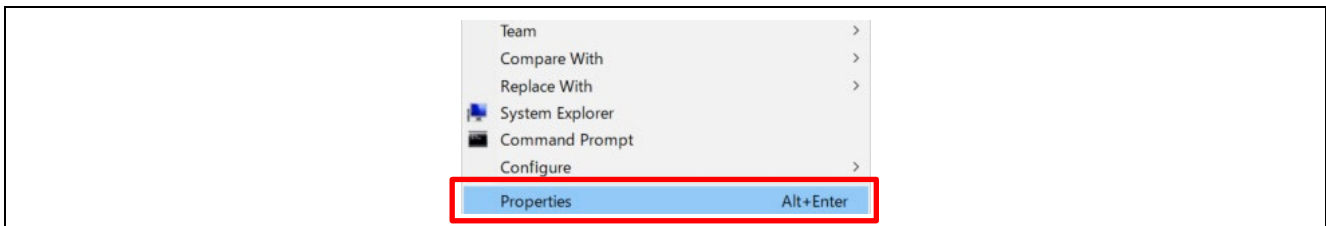


Figure 4-1 Context Menu of Project

- On the **Properties** dialog box, select **Builders** from the menu panel on the left. Then select **SC Code Generation Builder** and click the **Edit** button.

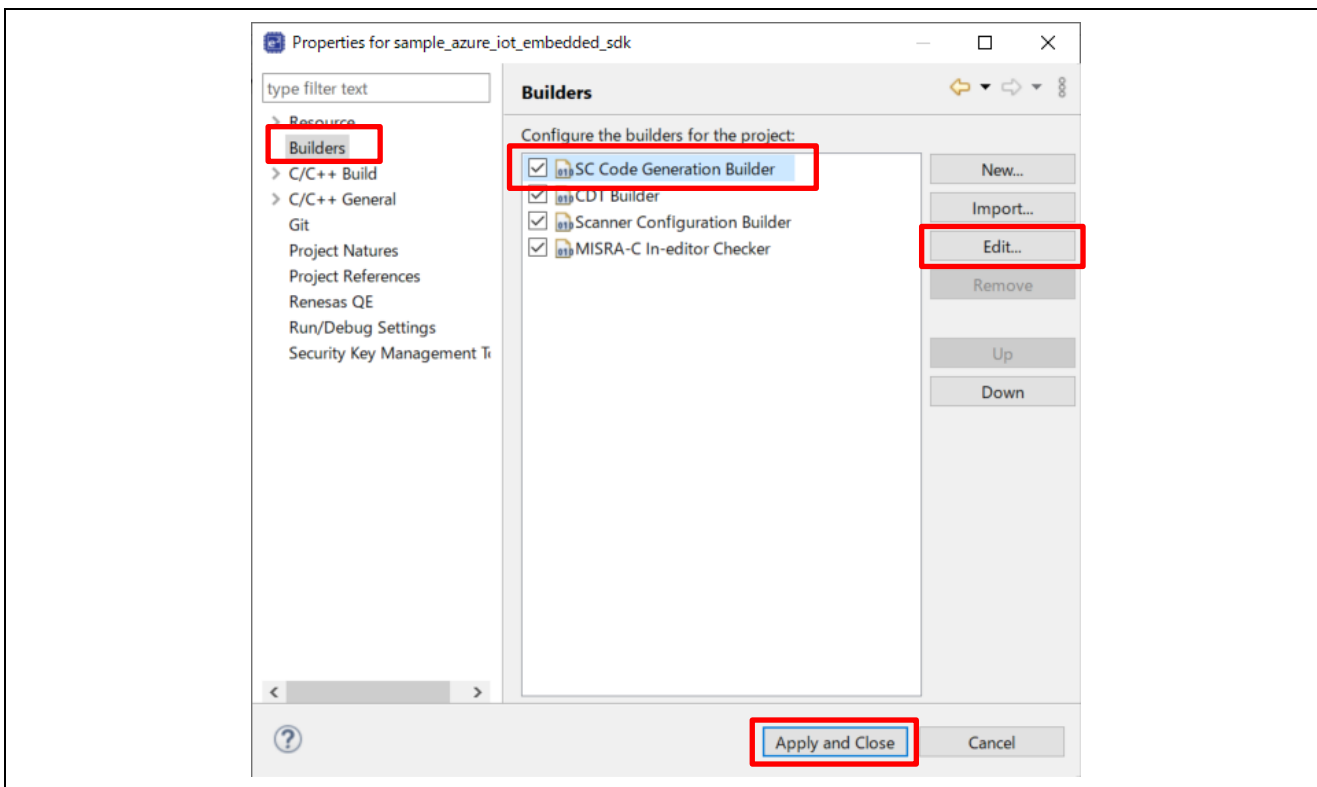


Figure 4-2 Properties Dialog Box of Project

- On the **Configure Builder** dialog box, uncheck all the boxes and click the **OK** button. Then click the **Apply and Close** button on the **Properties** dialog box. Configuring these settings will prevent unintended code generation.\*1

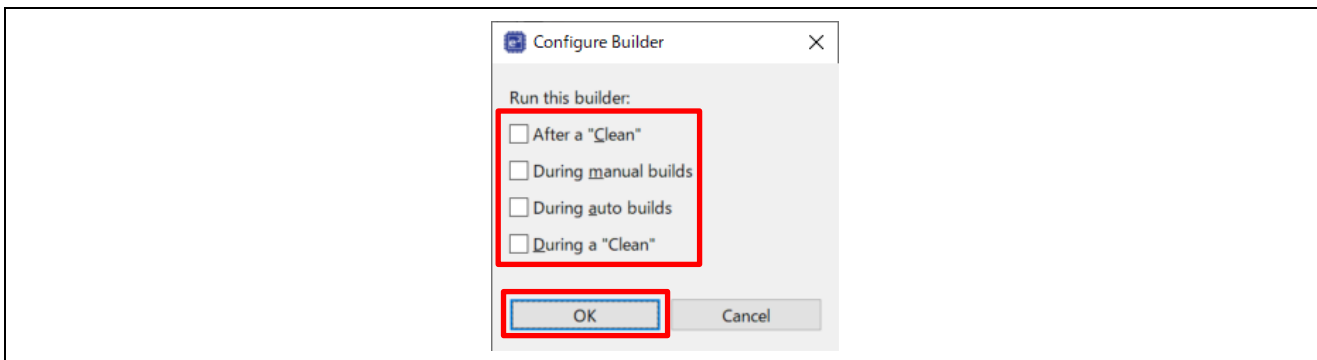


Figure 4-3 Configure Builder Dialog Box

Note: 1. If you actually want to trigger automatic code generation, click the **Generate Code** button in Smart Configurator.

## 4.2 Building the Project

Select **Project** → **Build All** to build the project. Note that a warning message appears at this time, but this does not indicate a problem.

After the build finishes, connect the target board to the PC and router as shown in Figure 2.1. Then select **Run** → **Debug** → **2 Renesas GDB Hardware Debugging** to start debugging.

Also, if you connect J12 on the CK-RX65N board to a PC with a USB cable, you can monitor the operating status using a terminal emulator program such as Tera Term. When you connect J12 to the PC, a port is registered in Windows Device Manager. Configure the settings of the terminal emulator program using the newly registered COM port number and connect to the target board. Configure the serial port communication settings as follows.

- Baud rate: 115,200 bps
- Data bits: 8
- Stop bits: 1
- Parity: None

When you run the project, device registration (provisioning) on the IoT hub is performed via a DPS connection. After registering the device on the IoT hub via the DPS connection, a connection is established to the IoT hub and a message is output via MQTT.

The sample program reports the operating status as serial output to the terminal emulator, as shown in the example screenshot below. Check the text displayed in the terminal emulator to confirm the following.

- [1] Successful connection to DPS
- [2] Successful connection to IoT hub by DPS
- [3] Successful connection to registered device ID

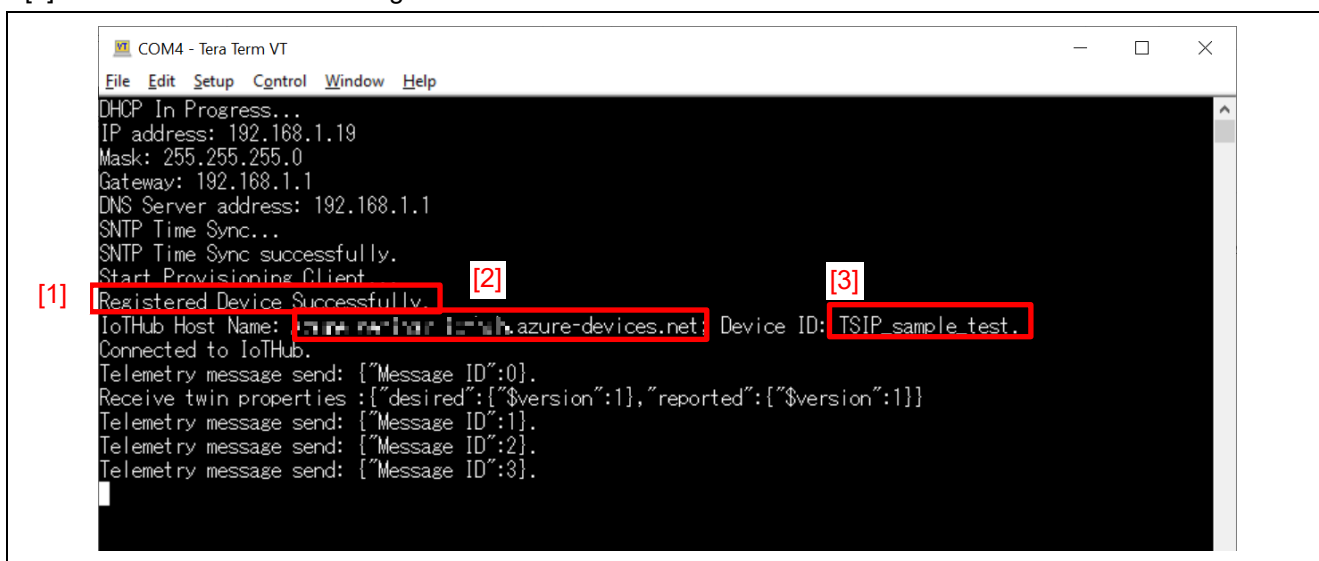


Figure 4-4 Example Output to Terminal Emulator

### 4.3 Confirming Connection to Microsoft Azure

You can confirm the connection to Microsoft Azure, and confirm that uploaded data was successfully sent to the Azure cloud, by using Azure IoT Explorer (preview). Refer to 3.5, Communication confirmation by Azure IoT Explorer in [Visualization of Sensor Data using RX65N Cloud Kit and Azure RTOS](#) for details.

In Azure IoT Explorer (preview), click < connection target IoT hub > → < registration ID >, then click **Telemetry** on the menu panel and click the **Start** button.\*1 If received data appears below **Receiving events...**, sending and receiving of messages is working properly.

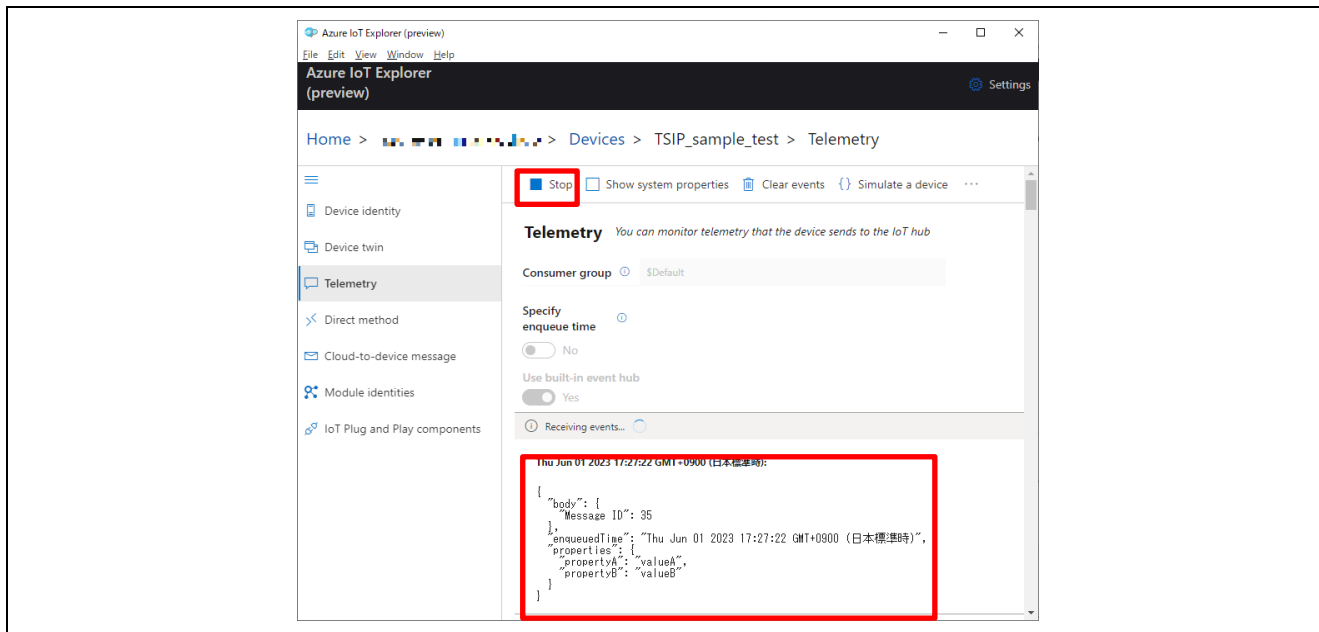


Figure 4-5 Confirming Reception of Telemetry Messages in Azure IoT Explorer

Note: 1. When reception starts after clicking the **Start** button, the display changes to a **Stop** button as shown in the screenshot.

#### 4.3.1 Checking Registration Status

To confirm that a device is successfully registered on the IoT hub via DPS connection, proceed as described below.

Select the connection target DPS in Azure Portal, click **Manage enrollments** on the **Settings** menu, and select the **Individual enrollments** tab to display a list of registration IDs. Click the connection target registration ID to display the **Enrollment details** page for the device.

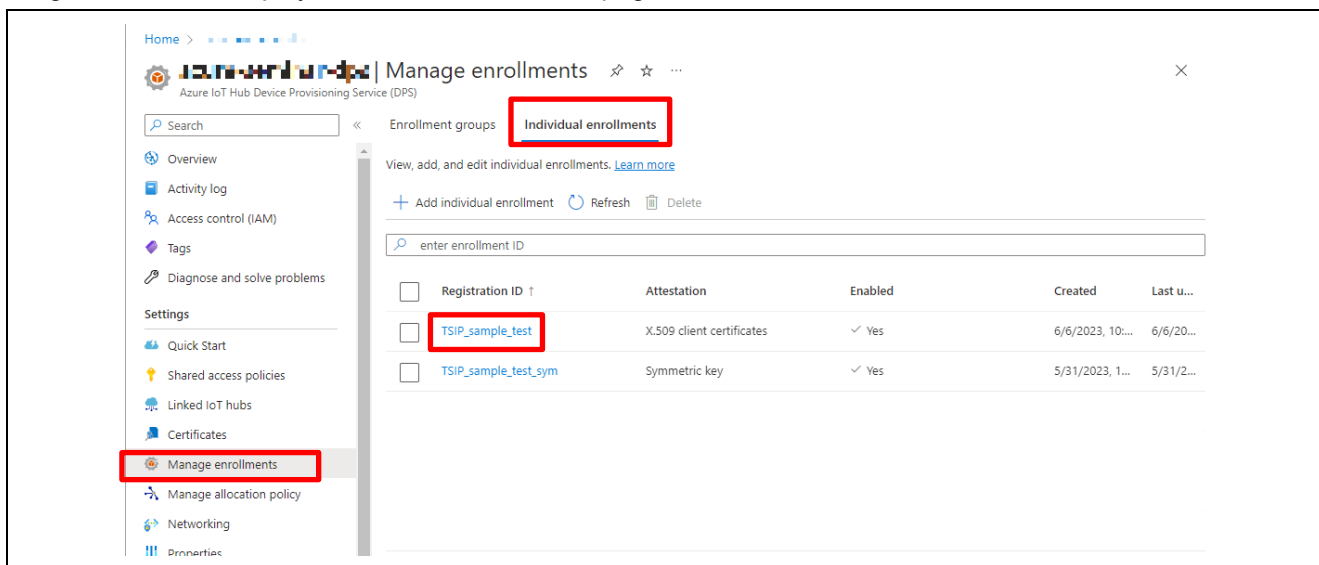


Figure 4-6 List of Registration IDs



## RX Family TLS Implementation Example Using TSIP Driver (Azure RTOS)

The registration status is displayed on the **Enrollment details** page for the device. Here you can confirm the IoT hub to which the DPS instance is assigned and the device ID.

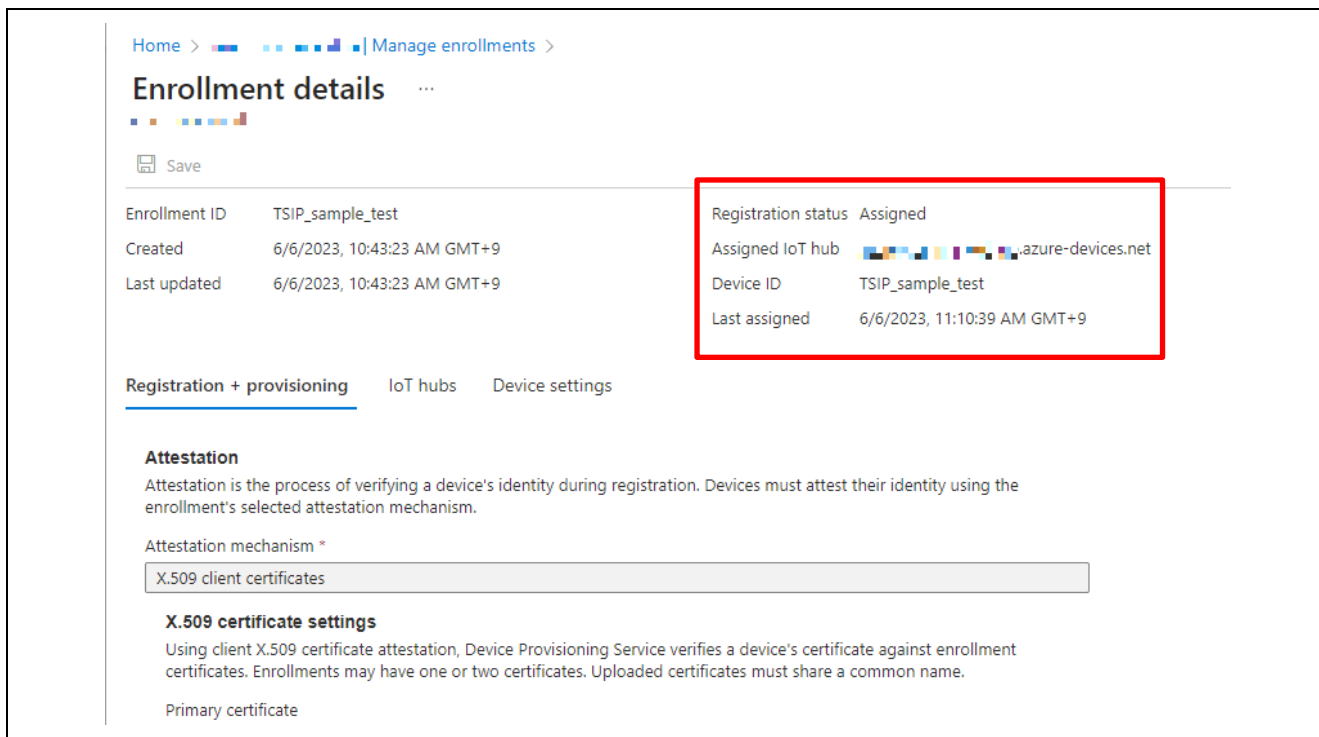


Figure 4-7 Enrollment details Page

### 4.3.2 Checking the Device

Confirm that the device has been added to the IoT hub registered on the DPS instance as described in 4.3.1. On the Azure Portal home page, select the IoT hub registered on the DPS instance, then click **Devices** under **Device management** on the menu panel. A list of devices is displayed, with the connected device newly added to it.\*1

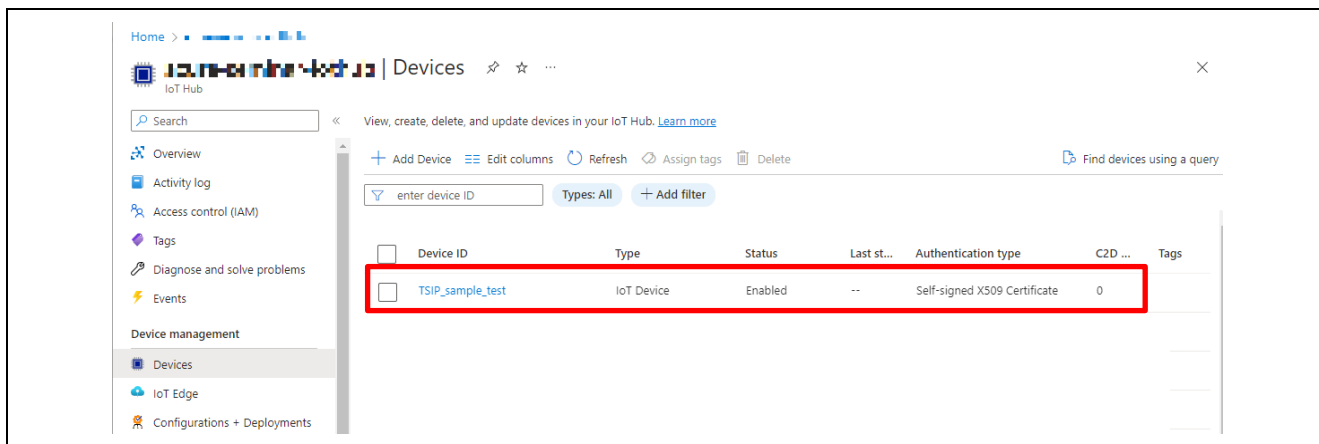


Figure 4-8 IoT Hub Device List

Note: 1. If the device has not connected even once it will not appear in the list.

## 5. Appendix

### 5.1 Details of the Security Key Management Tool

For details of the Security Key Management Tool, refer to the applicable document (downloadable from the following website):

<https://www.renesas.com/jp/ja/software-tool/security-key-management-tool>

### 5.2 TLS Communication Performance Using TSIP Driver

For reference, Table 5-1 lists examples of handshake times when establishing a TLS connection and application data transfer speeds after a TLS connection is established using the Renesas Starter Kit+ for RX65N-2MB. The MCU's internal timer was used to measure transfer times when uploading 4 KB of data and downloading 1 MB of data. Five sets of transfers were performed, and the average times were calculated. In these examples, using the TSIP driver reduced the handshake time when establishing a TLS connection from 2.73 seconds to 0.34 seconds, boosted the upload transfer rate from 4.46 Mbps to 24.82 Mbps, and boosted the download transfer rate from 5.34 Mbps to 27.03 Mbps.

**Table 5-1 Examples of TLS Communication Speeds Using TSIP Driver**

Cipher Suite	Block Cipher	NetX Duo* <sup>1</sup>	NetX Duo w/ TSIP* <sup>2</sup>
TLS_RSA_WITH_AES_128_CBC_SHA	128-bit AES-CBC	Connection: 2.73 seconds Up: 4.46 Mbps Down: 5.34 Mbps	Connection: 0.34 seconds Up: 24.82 Mbps Down: 27.03 Mbps

Notes: System clock (ICLK): 120 MHz

TSIP operating clock (PCLKB): 60 MHz

1. NetX Duo: Software processing
2. NetX Duo: with TSIP: Using TLS APIs of TSIP driver

## 6. Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 31, 2023	—	First edition issued.
1.10	Dec. 14, 2023	1,31	Added explanation for RX72N (Envision kit) compatibility.
		1,9,13	Update IDE/Azure RTOS/RDP/OpenSSL version.
		5	Modification of TLS flow with TSIP diagram
		14-19	Added instructions on obtaining the root CA certificate DigiCert Global Root G2.
		20	Added instructions for setting the root CA certificate to be used.
		24	Add DigiCert files to list.
		24,35	Added explanation of key data used for wrapping generated by script.
2.00	Jan. 23, 2024	1	Added the “Important Notice” section.
		1,2,8	Added descriptions with addition of target boards.
		2	Added a description of support for CK-RX65N+RYZ014A (Cellular), Renesas Starter Kit+ for RX65N-2MB, and CK-RX65N Cellular. (This document basically provides information about CK-RX65N.)
		2,9	Updated the version of IDE/Azure RTOS/RDP.
		2,5,7,9, 24,25,29, 30,34-41	Changed the key generation tool to the Security Key Management Tool.
		7,24,25, 29,30, 32-34	Modified terms of the key generation tool with the adoption of the Security Key Management Tool.
		10	Deleted the “tool” folder from the project folder because the distribution method was changed with the change of the key generation tool.
		19	Eliminated the need for setting or selecting the root CA certificate because all existing certificates are now verified.
		19	Added a description of the default files provided in the sample project so that root CA certificates can be registered.
		22	Added “DigiCertGlobalRootG2.cer” in the “ca” folder.
		23	Added a procedure for changing the file name of the root CA certificate in DER format.
		23,41,52	Edited paths because the folder that stores configuration files was changed to “userdata_tsip”.
		29,30, 34-40	Added a description of how to use the Security Key Management Tool.
		51,52	Added a description of the procedure for setting the access point when using the CK-RX65N+RYZ014A (Cellular) board.
		53	Deleted the NetXDuo Addons settings from the “nx_secure_port.h” file because they are now included in a patch file.
53	Added a procedure for checking the macro settings when using the Renesas Starter Kit+ for the RX65N-2MB board.		
58	Changed a sentence so that the user is informed of the documents that provide detailed information about how to use the Security Key Management Tool.		

## General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

### 1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

### 2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

### 3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

### 4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

### 5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

### 6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.).

### 7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

### 8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

## 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 incorporation 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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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; machine 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.Unless expressly 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; undersea 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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 subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please 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.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://www.renesas.com)

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:  
[www.renesas.com/contact/](http://www.renesas.com/contact/).