RA6W1 Getting Started with AWS IoT Core

The RA6W1 is a highly integrated ultra-low-power Wi-Fi® MCU that allows you to develop a complete Wi-Fi® solution on a single chip. This document is a RA6W1 manual intended to help new or existing developers quickly get started using AWS® IoT Core.

Contents

Figures

Tables

1. Terms and Definitions

AWS	Amazon Web Services
ADC	Analog to Digital Converter
AP	Access Point
AWS	Amazon Web Services
CA	Certificate Authority
DPM	Device Power Management
eMMC	embedded Multimedia Card
EVB	Evaluation Board
FreeRTOS	Free Real-Time Operating System
ID	Identifier
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
MUX	Multiplexer
NVRAM	Non-Volatile Random Access Memory
OS	Operating System
OTA	Over the Air
RF	Radio Frequency
RTC	Real-Time Clock
SD	Secure Digital
SDK	Software Development Kit
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
Wi-Fi	Wireless Fidelity

2. References

[1] RA6W1 Datasheet, Renesas Electronics.
[2] RA6W1Getting Started Guide, Manual, Renesas Electronics.
[3] RA6W1 Host Interface, Manual, Renesas Electronics.

Note 1 References are for the latest published version, unless otherwise indicated.

3. Overview

The RA6W1 Wi-Fi Development Kit provides a quick and easy method to start developing battery powered applications and products using ultra-low power Wi-Fi.

The ultra-low-power RA6W1 chipset is the world's lowest power Wi-Fi chip specifically designed to meet the requirements for power sensitive wireless applications.

Benefits

- Ultra-low-power Wi-Fi technology
- More than 1-year battery life for most applications
- Industry leading wireless range
- Fully integrated Wi-Fi MCU (Microcontroller Unit)
- Comprehensive security capabilities
- Easy-to-use development tools means shorter time to market

Features

- Highly integrated ultra-low-power RA6W1 Wi-Fi system module
- Best Radio Frequency performance
- SoC runs full networking OS and TCP/IP stack
- Built-in 4-channel auxiliary ADC for sensor interfaces
- Built-in hardware crypto engines for advanced security features
- Complete software stack
- eMMC/SD expanded memory

Applications

RA6W1 is a full offload MCU for IoT Applications, such as:

- Security systems
- Door locks
- Thermostats
- Garage door openers
- Blinds
- Lighting control
- Sprinkler systems
- Video camera security systems
- Smart appliances
- Video doorbell

4. Hardware Description

The RA6W1 is a fully integrated Wi-Fi® MCU with an ultra-low-power consumption, best RF performance, and easy development environment. For more details, see Ref. [2].

5. Set up Development Environment

AWS IoT applications can be developed for the RA6W1 using the RA6W1 FreeRTOS Software Development Kit (SDK) and the Renesas Electronics e^{2}studio IDE on either a Windows 10 or Linux-based development system.

To start the software development environment:

1. Install and configure the e^{2}studio IDE.
2. Import the RA6W1 SDK into the e^{2}studio and build an application.
3. Set up evaluation board.
4. Download and test the application.
5. Use J-Link debugger to debug the application.

For more information, see Section 7 of Ref. [2].

6. Build and Run Reference Application

To get started with AWS IoT connectivity on the RA6W1 device, begin with a ready-to-use reference application. The provided application project, rm_awsiot_w_aws_example_ek_ra6w1 includes pre-configured project settings, middleware, and example code, which helps to save time and ensures that the environment is correctly set up for AWS IoT development. It shows how we can create an application which implements a cloud-connected door lock system in RA6W1 using the AWS IoT platform. TheRA6W1 device connects to a Wi-Fi network and establishes a secure MQTT connection to AWS IoT Core. The actual door lock hardware is expected to be controlled by RA6W1. The mobile application communicates with AWS IoT Core through the internet to remotely control the door lock through RA6W1.

The following components shown in Figure 1 are required to run the AWS IoT reference application in RA6W1 through an Internet connection and AWS IoT server:

- AWS IoT reference application package
- RA6W1 EVB
- Router: Connection to internet
- Mobile device: Android/iOS application
- AWS account.

Figure 1. Architecture of AWS IoT reference application

6.1 Load AWS-IoT Reference Application in e^2 studio

For more information on how to load the template, see Section 7.5 of Ref. [2].

In the RA6W1 SDK, you can configure the following parameters of AWS IoT core which uniquely identify the AWS IoT account and the device itself:

- Edit Endpoint or Host Name
- Edit Thing Name
- Edit Root CA, Client Certificate, and Client Private key

6.1.1 Edit Endpoint or Host Name

To change the AWS IoT Host Name:

1. Go to **Stacks** > **Properties** of `rm_awsiot_w stack`.
2. Enter the new value in **AWSIOT_W Host Name**.
3. Click **Generate Project Content in Stacks**. This action updates the macro AWS_IOT_HOST_NAME in the `rm_awsiot_w_cfg.h` file.

The macro is the following:

```
#define AWS_IOT_HOST_NAME     "(account-specific-prefix).iot.(aws-region).amazonaws.com
```

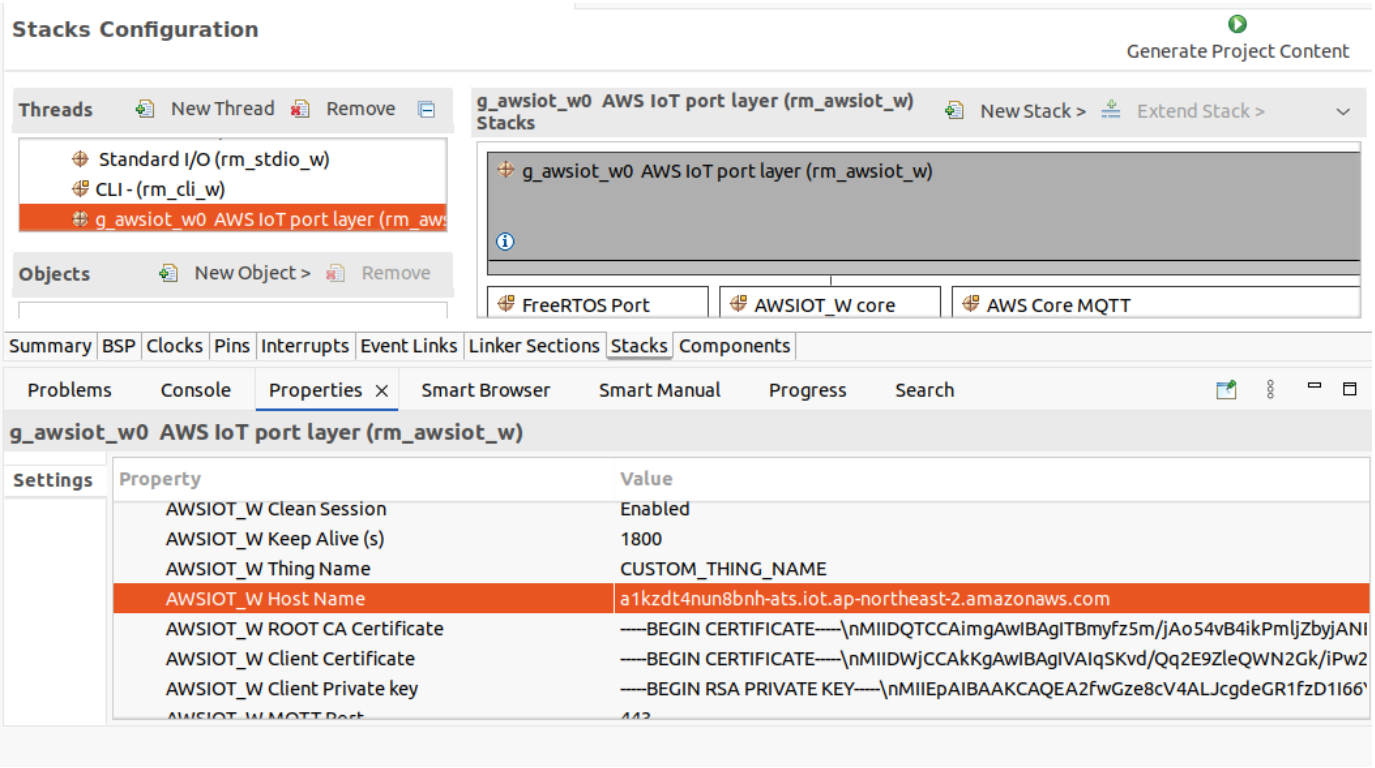

Figure 2. AWSIOT Host Name change in e^{2}studio

6.1.2 Edit Thing Name

If the Thing Name does not exist in NVRAM, the system uses the predefined name from the configuration header file.

To change the AWS IoT Thing Name:

1. Go to **Stacks** > **Properties** of **rm_awsiot_w stack**.
2. Enter the new value in **AWSIOT_W Thing Name**.
3. Click **Generate Project Content**. This action updates the macro AWS_IOT_THING_NAME in the `rm_awsiot_w_cfg.h` file, see Figure 3.

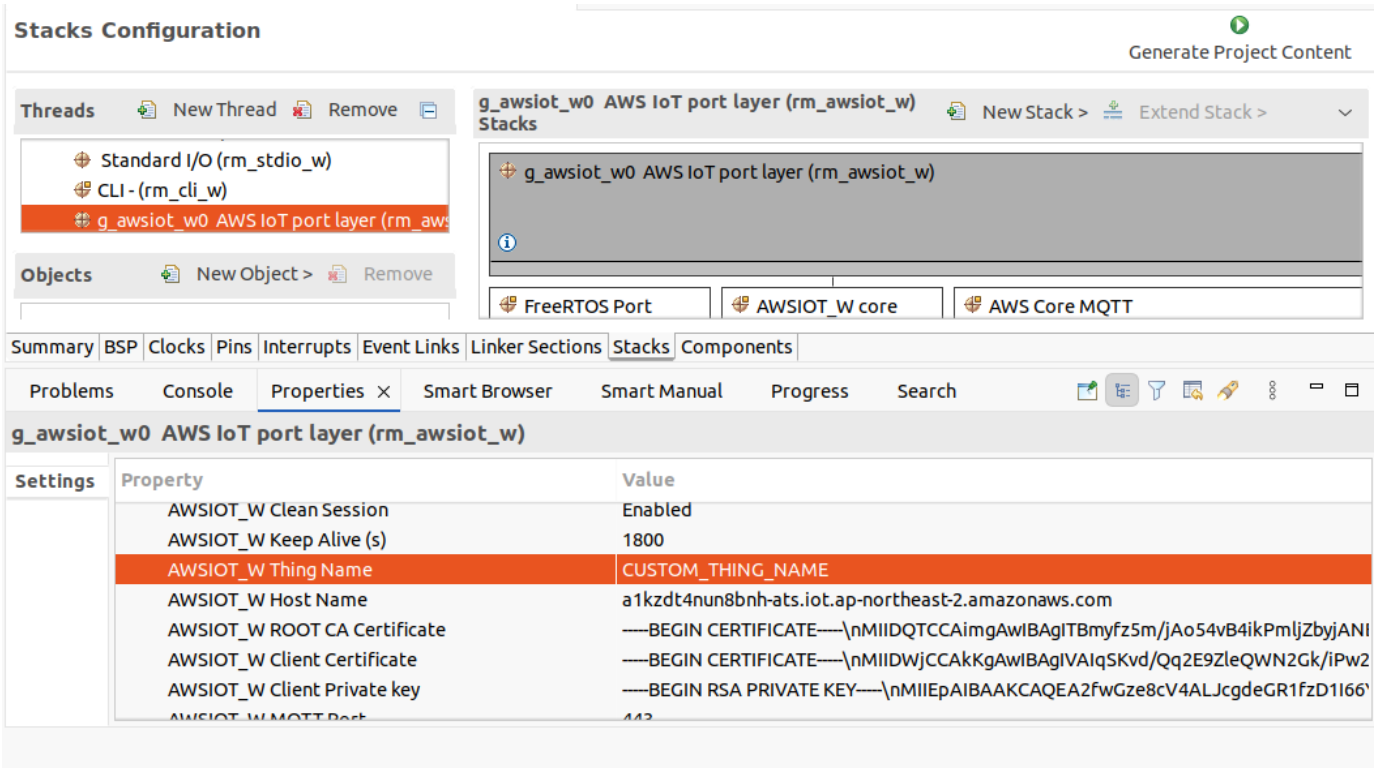

Figure 3. AWSIOT Thing Name change in e²studio

6.1.3 Edit Root CA, Certificate and Private Key

To authenticate the device with AWS IoT, the device must contain the Root CA, Client Certificate, and Client Private key. For more details, see https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html.

To update "AWSIOT_W ROOT CA Certificate", “AWSIOT_W Client Certificate” and "AWSIOT_W Client Private key" :

1. Go to **Stacks** > **Properties** for the `rm_awsiot_w stack`.
2. Click **Generate Project Content** in Stacks. This action updates the corresponding macros in the `rm_awsiot_w_cfg.h` file, see Figure 4.

```
#define #define AWS_IOT_ROOT_CA    "-----BEGIN CERTIFICATE-----\nMIIDQTCCAimgAwIBAgITBmy-
fz5m……rqXRfboQnoZsG4q5WTP468SQvvG5\n-----END CERTIFICATE-----\n"
#define AWS_IOT_CLIENT_CERT        “-----BEGIN CERTIFICATE-----
\nMIIDWjCCAkKgAwIBAgIVAIqSKvd……CO4wes8RH5pNIOK2QrKgr9NJkA==\n-----END CERTIFICATE-----\n"
#define AWS_IOT_CLIENT_PRIV_KEY     “"-----BEGIN RSA PRIVATE KEY-----
\nMIIEpAIBAAKCAQEA2fwGze8cV4A……OcWdbf54nhzcEqjRv1DhCA==\n-----END RSA PRIVATE KEY-----\n"
```

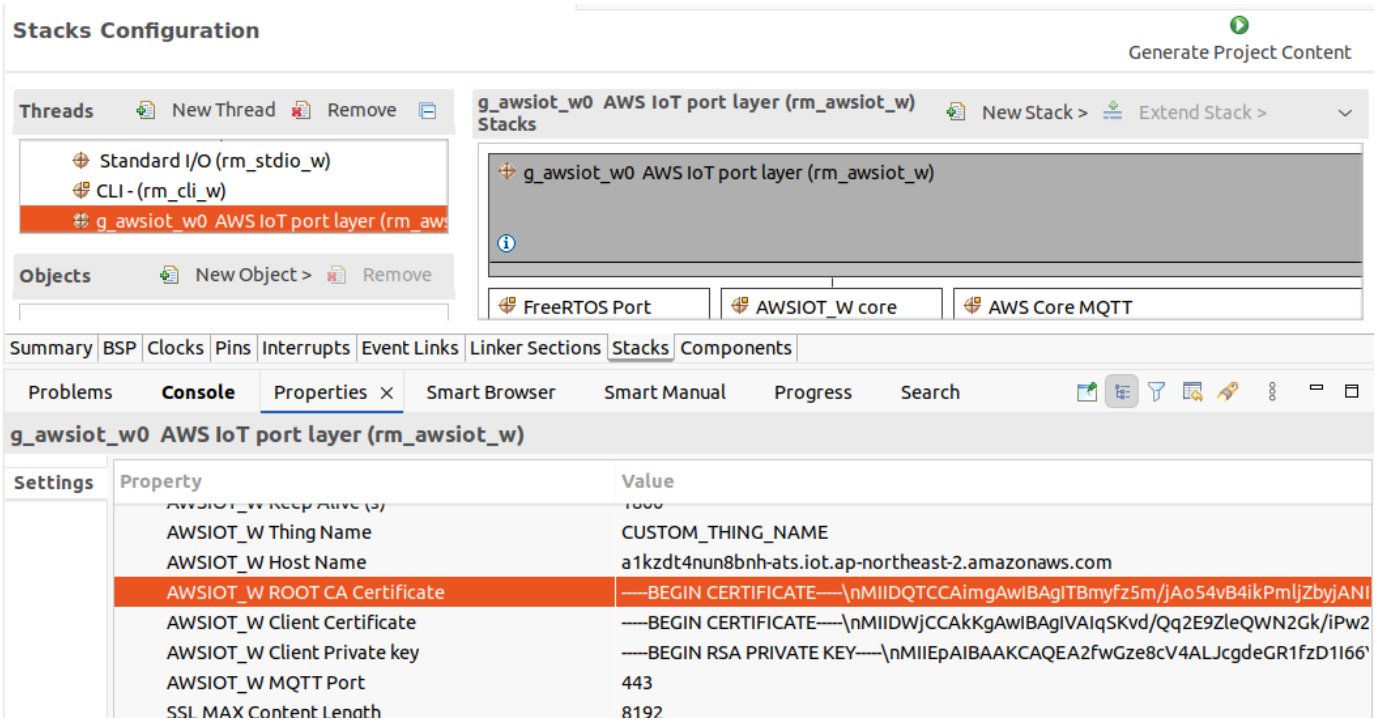

Figure 4. CA Cert, Client Certificate and Private key change in e^{2}studio

6.2 Build and Flash the Image

To flash the image, see Section 5 of Ref. [2].

6.3 Provision the Wi-Fi in RA6W1

Provisioning a Wi-Fi device is the process of setting the device up to connect to a network and function properly. This involves configuring network settings, ensuring secure authentication, and assigning necessary parameters. The provisioning can be done with the Renesas Wi-Fi Provisioning app on either an Android or iOS device.

When RA6W1 initially starts after flashing the image, it is in Station mode. To change the device into Provisioning mode and to do the Wi-Fi provisioning follow the steps:

1. Configure factory reset button, see Figure 5.
   If you want to use the button BTN1 for factory reset, connect the pins P0_10 and BTN1 together using a jumper wire.

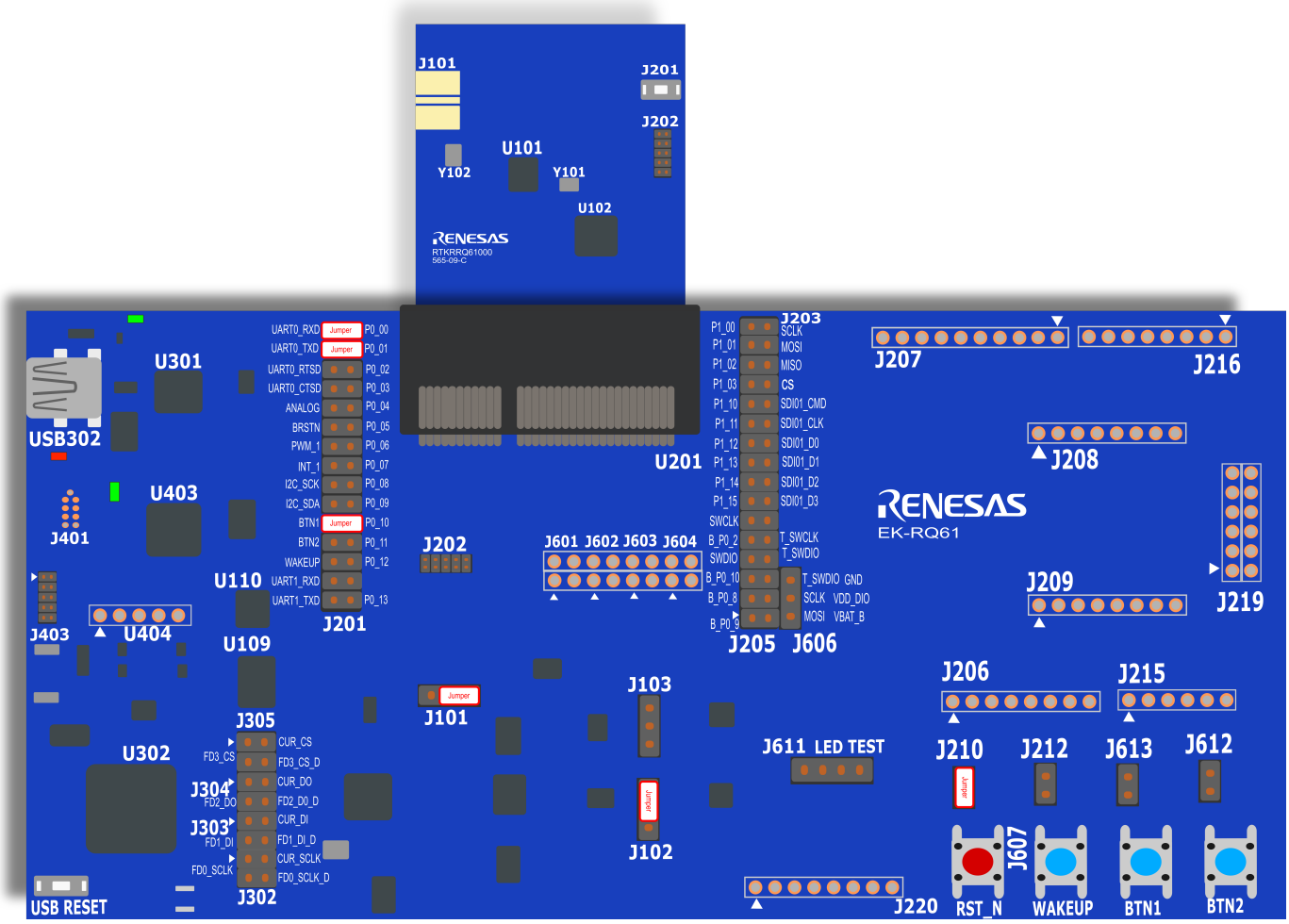

Figure 5. Configure factory reset button in the EVB for Wi-Fi Provisioning mode

2. Press the factory reset button configured using the jumper connection.
   Now the device does the factory reset and booting in Soft-AP mode and is ready for Wi-Fi provisioning.
3. Download the Renesas Wi-Fi Provisioning application from the Google Play Store or the Apple App Store to connect RA6W1 to Access Point, for Android, see Figure 6.
4. Open the application and click **While using the app** if the location access permissions are asked when you open the application for the first time.
5. Select **Start DA16200/RRQ61000-based**.
6. Click **I'm read** and click **Connect**.
7. Click **NEXT** when the application connects to RA6W1.
   Now the application lists all the Access Points RA6W1 scanned.
8. Select the needed Access Point and enter the password.
   This writes the Wi-Fi network details to the device.
9. In the success dialog, click OK.

Wi-Fi provisioning is now completed and RA6W1 reboots and starts as a Station and connects to the Access Point configured from the application. After the successful connection to the internet, it connects to the AWS server.

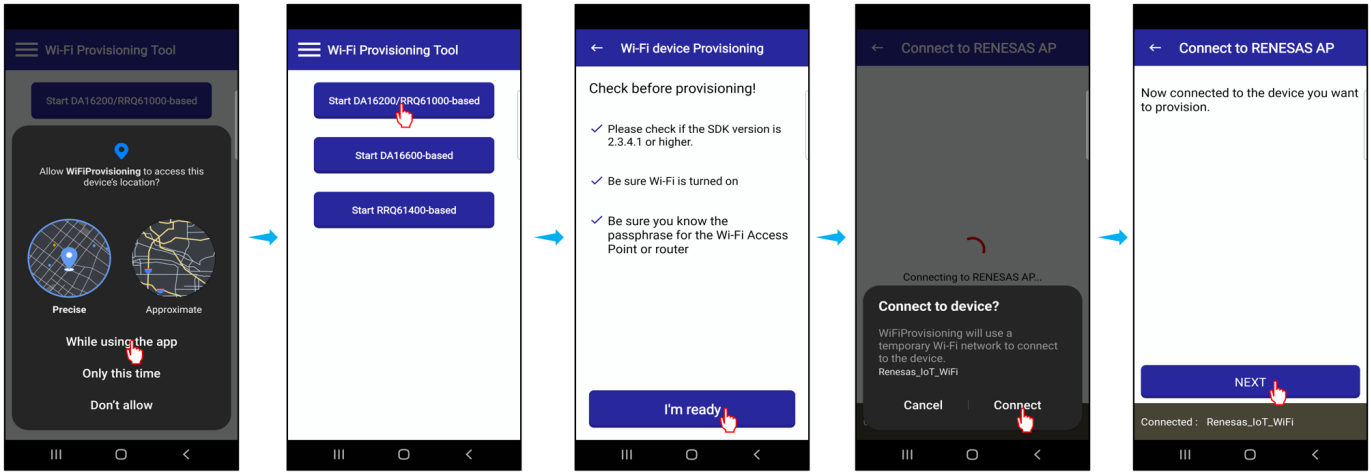

Figure 6. Renesas Wi-Fi provisioning application steps

6.4 Reference Application in RA6W1

When provisioning is completed, to open AWS application on mobile device, select **AWS IoT** by clicking the menu icon, see Figure 7, for details see Section 8 AWS IoT.

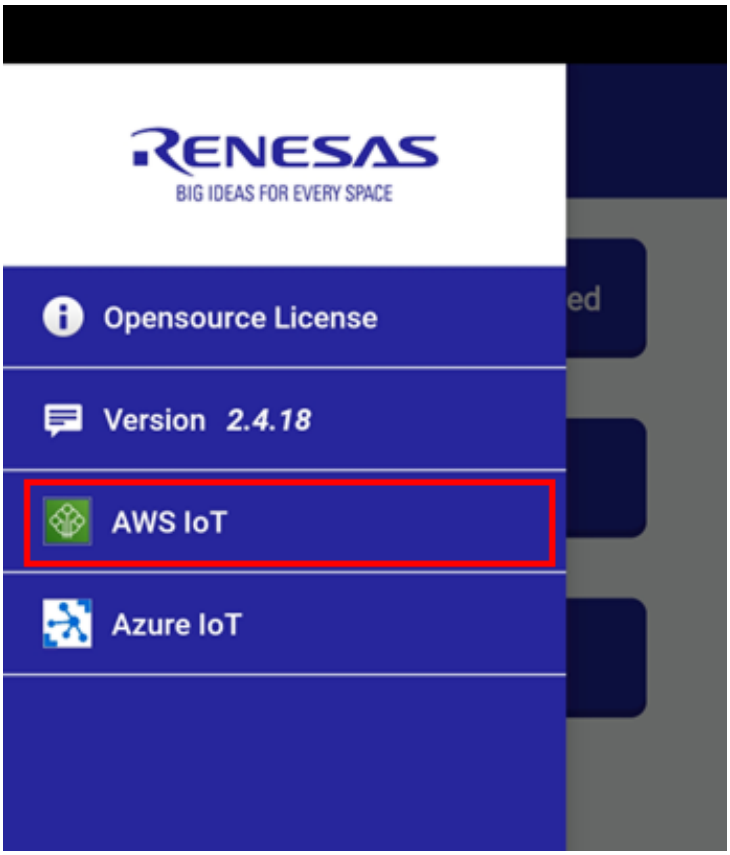

Figure 7. AWS IoT mobile application

In AWS IoT application you can close and open the door, by clicking the Lock icon, see Figure 8.

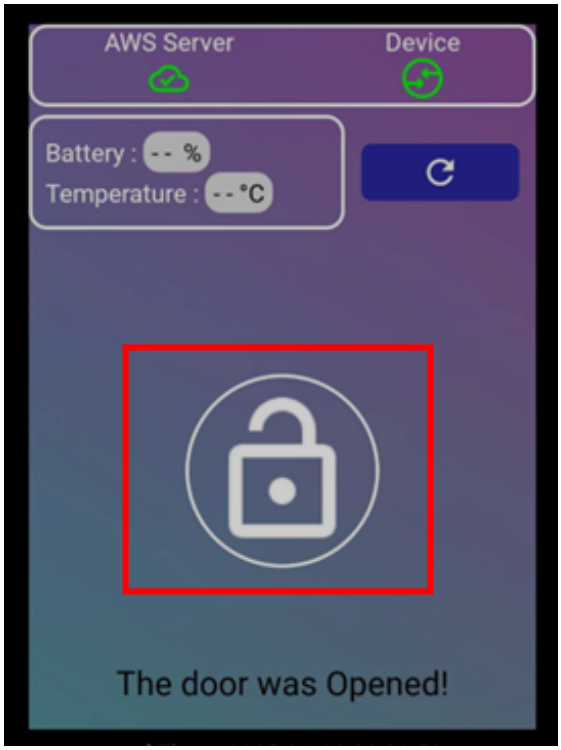

Figure 8. Open the door

Figure 9 shows message flow of opening the door.

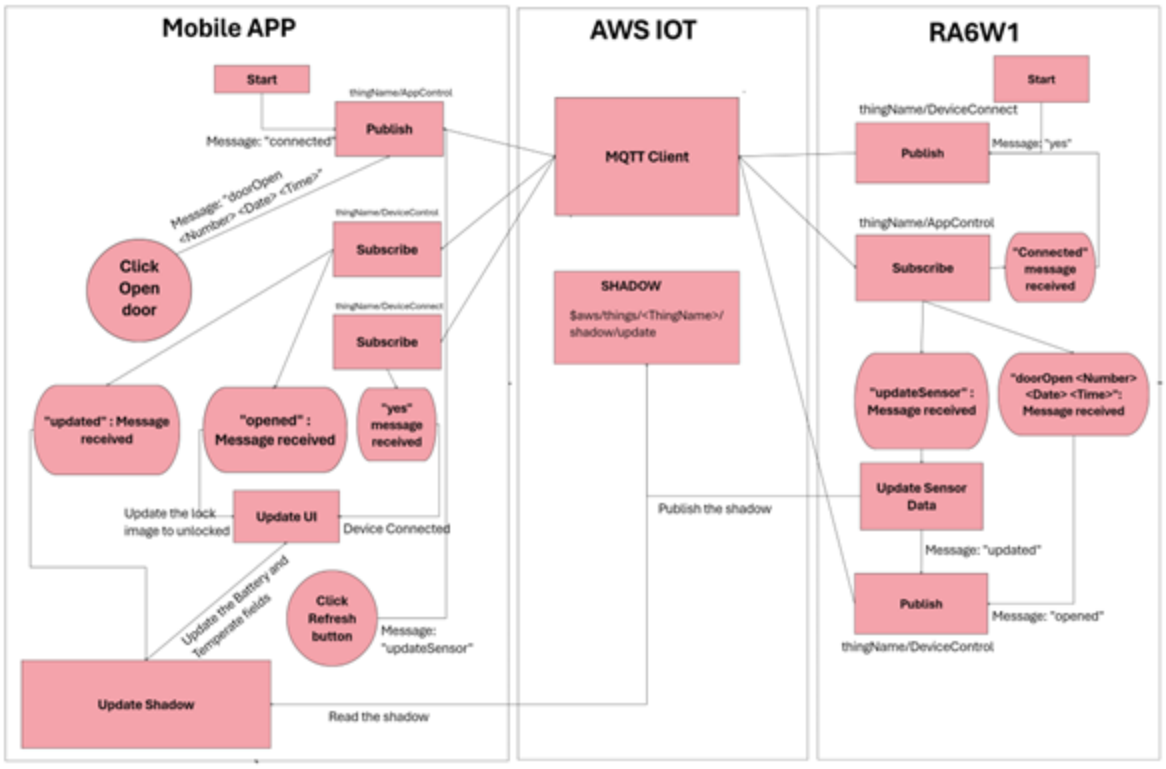

Figure 9. Message flows of opening the door

Figure 10 shows the operation of opening the door on Android app.

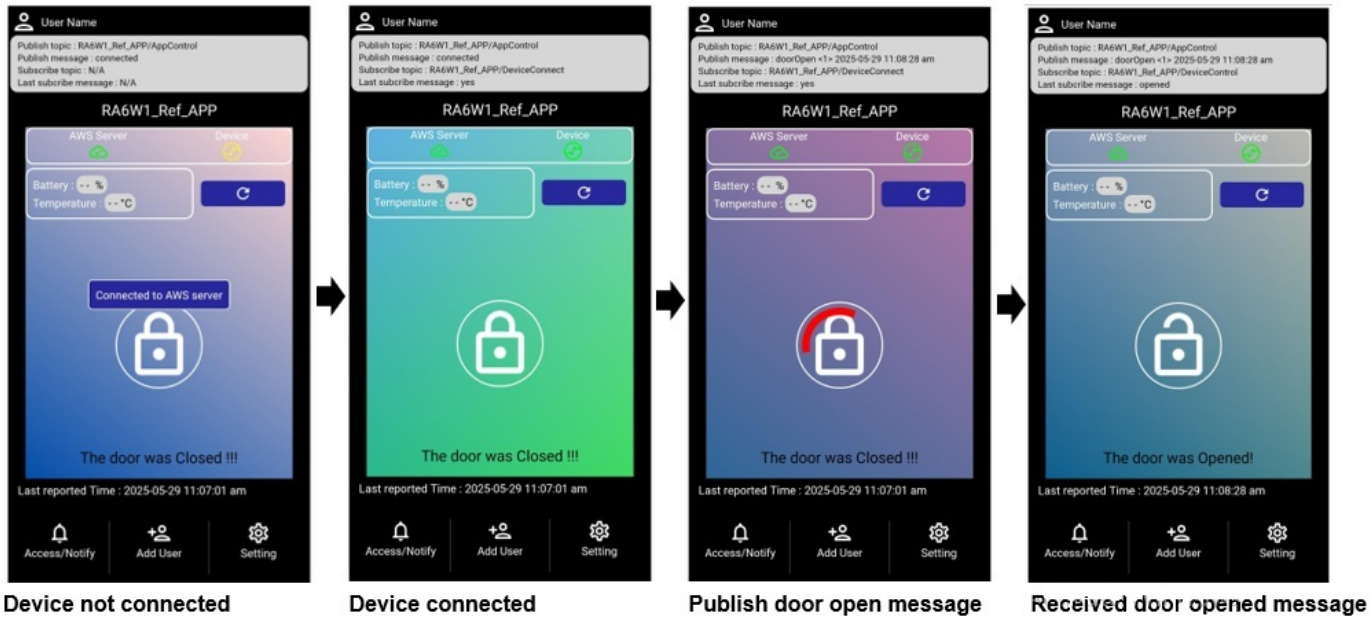

Figure 10. Opening the door on Android app

When the operation of opening the door is completed, the console logs of the RA6W1 appear as follows:

```
open comm
[openControl]

DEBUG:   [aws_dpm_app_door_work:1961] previous MQTT result = 0, doorLock CMD (=1: 0-idle, 1-open,
2-close, 3-auto close)
================================================================================
********************************************************************************
last user Timer ID = 5
last doorOpenFlag state: "true"
last FOTA Stat: 0
last FOTA Url: ""

Sleep mode DPM: KA timer interval(=1800 sec)

[DPM_App_Main] DM_FINISH_DEVICE
recv timeout(=19 ms) set OK (socket=0)
[dpm_keepalive_timer_unregister] OK to deregister AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to delete AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to register timer: interval = 1770(sec), tid = 5
>>> Start DPM Power-Down !!!
```

Figure 11 shows message flows of closing the door.

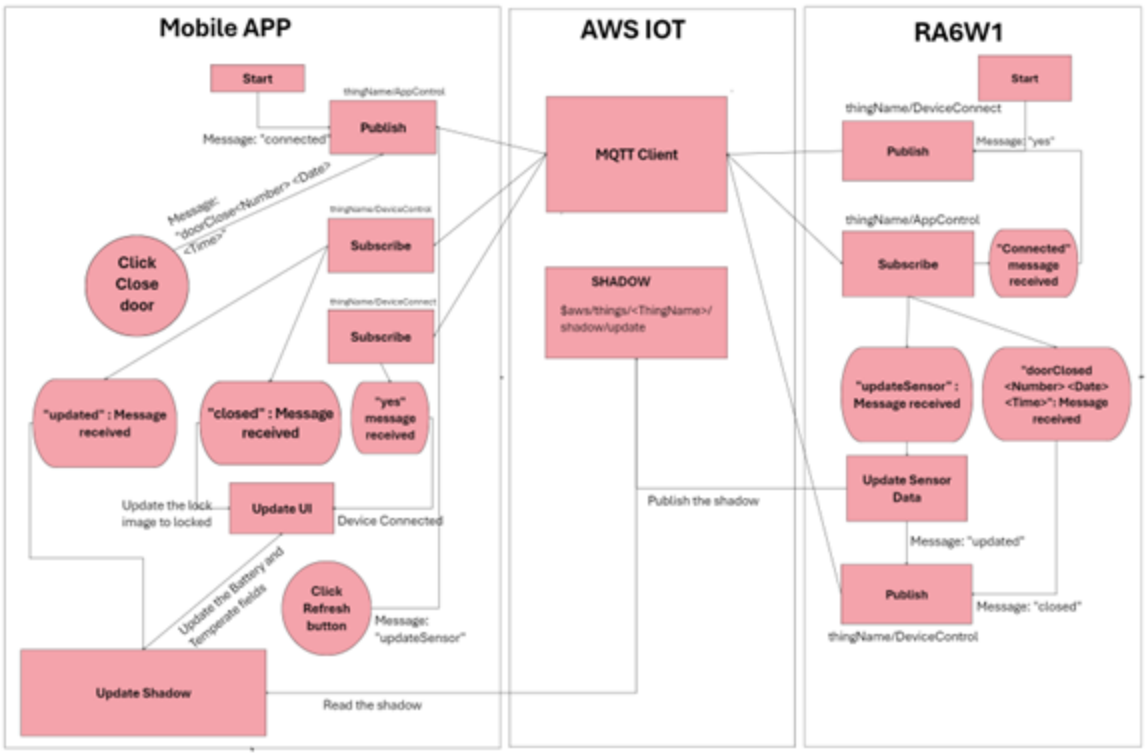

Figure 11. Message flows of closing the door

Figure 12 shows the operation of closing the door on Android app.

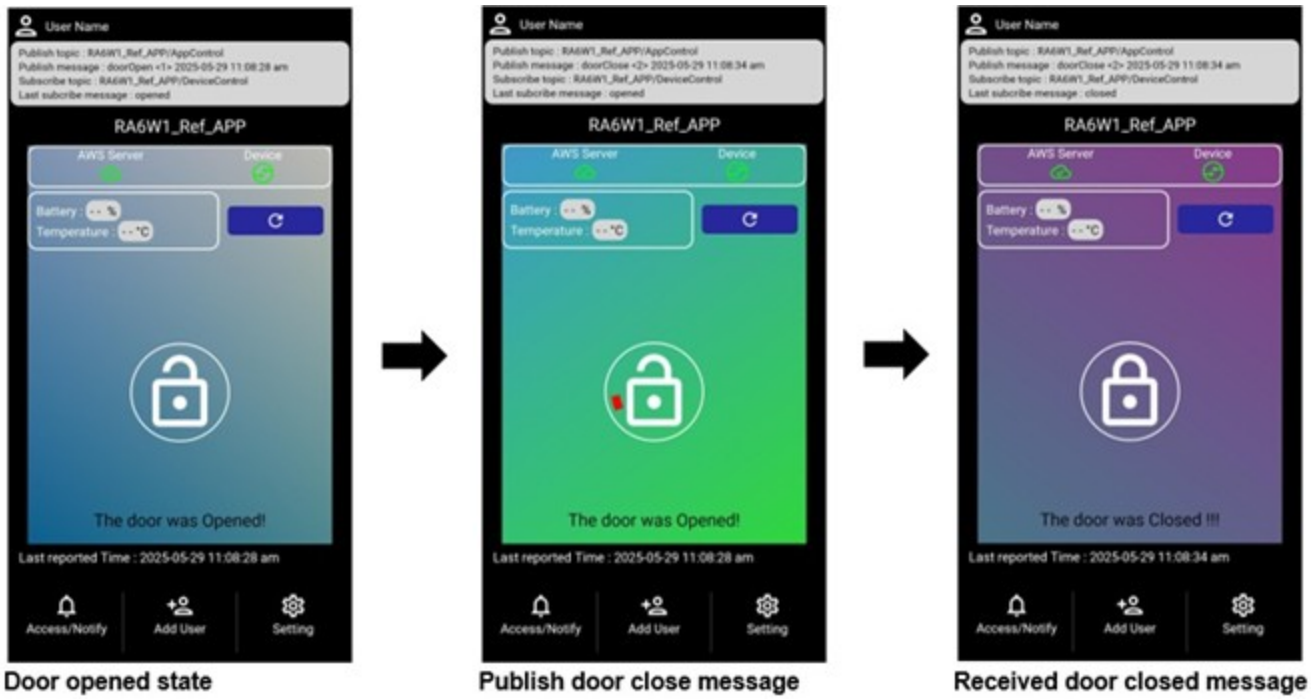

Figure 12. Closing the door on mobile app

When the operation of closing the door is completed, the console logs of the RA6W1 appears as follows:

```
close comm
[closeControl]
DEBUG:   [aws_dpm_app_door_work:1961] previous MQTT result = 0, doorLock CMD (=2: 0-idle, 1-open,
2-close, 3-auto close)
================================================================================
********************************************************************************
last user Timer ID = 5
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url: ""
Sleep mode DPM: KA timer interval(=1800 sec)
[DPM_App_Main] DM_FINISH_DEVICE
recv timeout(=19 ms) set OK (socket=0)
[dpm_keepalive_timer_unregister] OK to deregister AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to delete AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to register timer: interval = 1770(sec), tid = 5
[aws_dpm_app_finish_loop] break finish_loop...
>>> Start DPM Power-Down !!!
```

7. Reference Application using AT Commands

To get started with AWS IoT connectivity on the RA6W1 device which is using AT commands as an interface with external MCU, begin with a ready-to-use reference application. The provided application project rm_awsiot_w_aws_atcmd_example_ek_ra6w1 includes pre-configured project settings, middleware, and example code, which helps save time and ensures that the environment is correctly set up for development. It shows how we can create an application which implements a cloud-connected door lock system in RA6W1 where the door lock hardware relates to an external MCU which is interfaced with RA6W1 using UART – AT commands. The mobile application communicates with AWS IoT Core through the internet to remotely control the door lock through RA6W1 and MCU.

If external MCU is not available, you can also use the provided `ttl` scripts to test the reference application. The script replicates the AT commands from external MCU. You can find the scripts are in the following sections.

Figure 13 shows the components required to run the reference application:

- AWS IoT - ATCMD reference application package
- RA6W1 EVB
- External MCU EVB (RA6M4)
- Terminal application which can run `ttl` scripts (If external MCU is not used)
- Router: Connection to internet
- Mobile device: Android/iOS application
- AWS account

To get more information on the jumper configurations on RA6W1 EVB for AT, see Section 5.2.3.1 of Ref. [2].

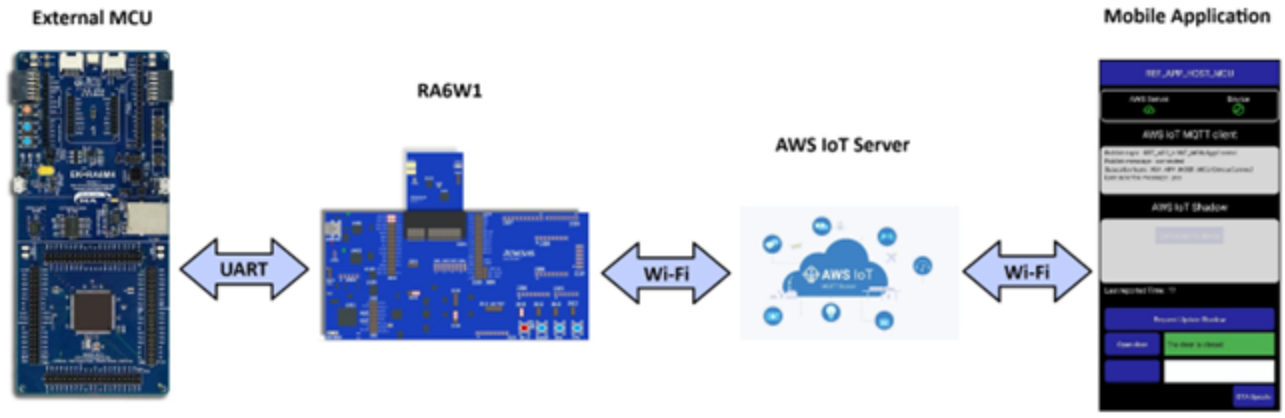

Figure 13. Architecture of AWS IoT reference application using AT commands

7.1 Load AWS-IoT Reference Application in e^2 studio

For more information on how to load the template, see Section 7.5 of Ref. [2].

7.2 Build and Flash the Image

To know about how to flash the image, see Section 5 of Ref. [2].

7.3 Provision the Wi-Fi and Configure AWS IoT Parameters

Follow step 1 and step 2 of the Section 6 Build and Run Reference Application, and then configure the following AWS IoT parameters:

- Unique Thing Name
- AWS Broker URL
- Publish and subscribe topic
- Root CA, Certificate and Private Key
- Thing attributes

These parameters can be set using the following `ttl` script.

```
; Wait for entering in provisioning mode
wait '+ATPROV=STATUS 1'
; Configure Thing name
sendln 'AT+AWS=SET,APP_THINGNAME,RA6W1_THINGNAME'
; Wait for ‘OK’ response
wait 'OK'
; Configure Broker URL
sendln 'AT+AWS=SET,AWS_BROKER,a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com'
wait 'OK'
; Configure Subscribe topic
sendln 'AT+AWS=SET,APP_SUBTOPIC,AppControl'
wait 'OK'
; Configure Publish topic
sendln 'AT+AWS=SET,APP_PUBTOPIC,DeviceControl'
wait 'OK'
; Configure Attributes
; Attribute app_door is used in the publish message from application
sendln 'AT+AWS=CFG 0 app_door 1 2'
wait 'OK'
; Attribute app_shadow is used in the shadow update publish message from application
sendln 'AT+AWS=CFG 1 app_shadow 1 2'
wait 'OK'
; Attribute doorStat is used in the device shadow update
sendln 'AT+AWS=CFG 2 doorStat 1 1'
wait 'OK'
; Attribute battery is used in the device shadow update
sendln 'AT+AWS=CFG 3 battery 2 1'
wait 'OK'
; Attribute DeviceControl is used to publish message to topic, DeviceControl
sendln ‘AT+AWS=CFG 4 DeviceControl 1 0’
wait 'OK'
; Enable DPM
sendln 'AT+AWS=SET,USE_DPM,1'
wait 'OK'
;; Configure RootCA Certificate
sendln #27 'C0,-----BEGIN CERTIFICATE-----'
sendln 'MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF'
sendln 'rqXRfboQnoZsG4q5WTP468SQvvG5'
sendln '-----END CERTIFICATE-----'
send #3
waitln 'OK'
MPause 500
;; Configure Client Certificate
sendln #27 'C1,-----BEGIN CERTIFICATE-----'
sendln 'MIIDWjCCAkKgAwIBAgIVAKQVbt137DPvbOJzpLZM84OFuaFhMA0GCSqGSIb3DQEB'
sendln 'wcESdbbU8mdKapYP3szjT1o97fji/Sh/Fhk7TB5vA561JED/8SvhmQk5HykFug=='
sendln '-----END CERTIFICATE-----'
send #3
waitln 'OK'
```

```
MPause 500
;; Configure Client Private Key
sendln #27 'C2,-----BEGIN RSA PRIVATE KEY-----'
sendln 'MIIEpAIBAAKCAQEAseR71M6gPGI3ljQhklUdo1gamQVjYhEENH4D/mxt3k4YTwY3'
sendln 'wXQBS62JONf9Vog2f6Z1yjspl5W7YmO8xwyEduRkRoY0zzS77QS06w=='
sendln '-----END RSA PRIVATE KEY-----'
send #3
waitln 'OK'
```

If external MCU is used instead of the `ttl` script, then the MCU should use the same AT commands shown in the source code.

After configuring the AWS IoT parameters, follow the rest of the steps (from step 3 to 10) in Section 6 Build and Run Reference Application for completing Wi-Fi provisioning.

7.4 Reference Application in RA6W1 using AT Commands

When provisioning is completed, to open AWS application on mobile device, select **AWS IoT** by clicking the menu icon.

- To open and close the door, click the **Open door**/**Close door** button, see Figure 14. The left side of Figure 14 shows the closed door state, and the right side show the opened door state.
- To update the shadow information, click **Request Update Shadow**. The shadow information from RA6W1 appears above the button.

Figure 14. Application for AWS IoT with AT command

When RA6W1 boots up, it connects to the provisioned Wi-Fi Access Point. After successful connection to internet, RA6W1 application connects to AWS server after fetching the configured AWS IoT parameters.
If the mobile application successfully connects to AWS server, the AWS server icon turns solid green. When the mobile application receives the message, "yes" on the subscribed topic, <Thingname>/DeviceConnect, the Device icon turns into solid green.

To collect the door lock state from external MCU, RA6W1 sends an AT command, +AWSIOT=CMD_TO_MCU update. The MCU should now respond with the shadow update, and it updates on the application. For example: AT+AWS-S=CMD MCU_DATA 2 doorStat closed 3 battery 80.

Now the initial connection of MCU, RA6W1, and mobile application is completed. If the device does not respond with the shadow update, then application notifies you that the device is not responding.

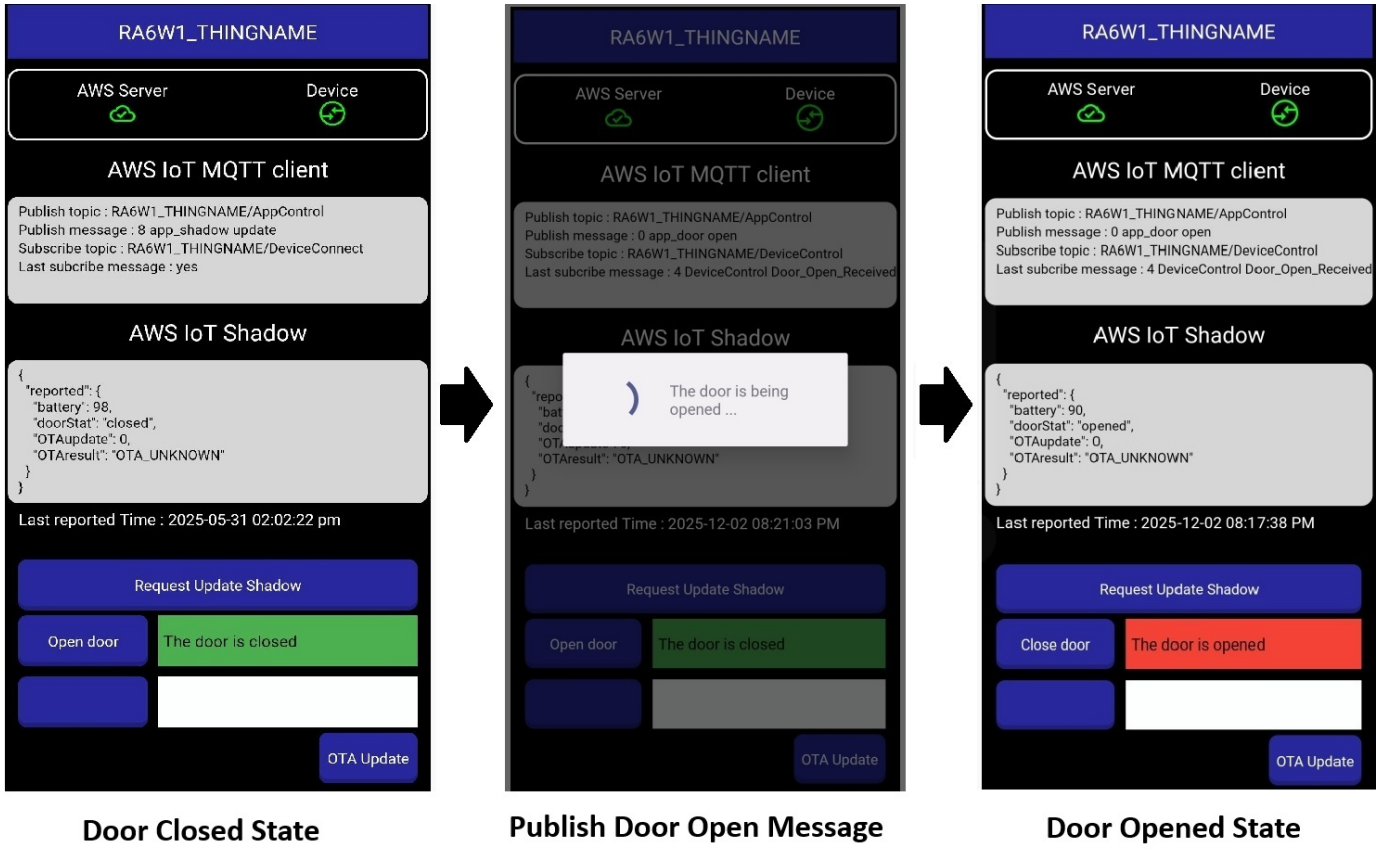

Figure 15. Opening door on mobile app with host MCU

The flow of opening the door door is the following, see Figure 15:

1. When you click **Open door**, application publishes message: 0 app_door open on the topic <Thing-name>/AppControl.
2. If the DPM is enabled, when RA6W1 receives the message, it sends an AT command, +INIT:WAKEUP,UC. MCU should now keep the DPM from going to sleep by sending the command AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,4. Now RA6W1 sends the command +AWSIOT=SERVER_DATA 0 app_door open and waits for response until timeout.
3. If the DPM is not enabled, then RA6W1 directly sends the command +AWSIOT=SERVER_DATA 0 app_door open and wait for response until timeout.
4. MCU should respond with the command: AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, as the application is subscribing to the topic <Thing-name>/DeviceControl.
5. MCU should now process the command and do the door opening. RA6W1 now again sends a command +AWSIOT=CMD_TO_MCU update to get the status from MCU and waits for response until timeout.
6. Now MCU updates the shadow with the command, AT+AWS=CMD MCU_DATA 2 doorStat opened 3 battery 90.
7. RA6W11 parses this and publishes the shadow to the application and the box under AWS IoT Shadow is updated according to the received values also, the state of door is changed in the application along with the shadow.

Console log for opening the door is the following:

```
pal_app_event_cb 0 app_door opene
dynamic subscription command ("app_door")
send "SERVER_DATA 0 app_door open" to MCU
=======================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 4 DeviceControl Door_Open_Received
=======================================================
topicCount : 1
Count : 0, cmdNum = 4
mqtttype = 0
data type(publish)=1
call publish: 4 DeviceControl Door_Open_Received
publish (contents update) OK - topic : RA6W1_THINGNAME/DeviceControl payload: 4 DeviceControl Door_
Open_Received
release response
*******************************************************************************
send "CMD_TO_MCU update" to MCU
=======================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 2 doorStat opened 3 battery 90
=======================================================
topicCount :
Count : 0, cmdNum = 2
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = opened
Count : 1, cmdNum = 3
mqtttype = 1
index(=0) matched
data type(shadow) = 2
call update sensor(need to be set variable): battery = 90.000000
release response
*******************************************************************************
publish (shadow sensor update) OK -  payload:"{"state":{"reported":
{"battery":90.000000,"doorStat":"opened","OTAupdate":0,"OTAresult":"OTA_UNKNOWN"}},"cli-
entToken":"RA6W1_THINGNAME-0"}"
last user Timer ID = 0
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url:""
```

AT console log for opening the door is the following:

```
+AWSIOT=SERVER_DATA 0 app_door open
>>>>> command: <AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Open_Received> is given here <<<<<
OK
+AWSIOT=CMD_TO_MCU update
>>>>> command: <AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80> is given here <<<<<
OK
```

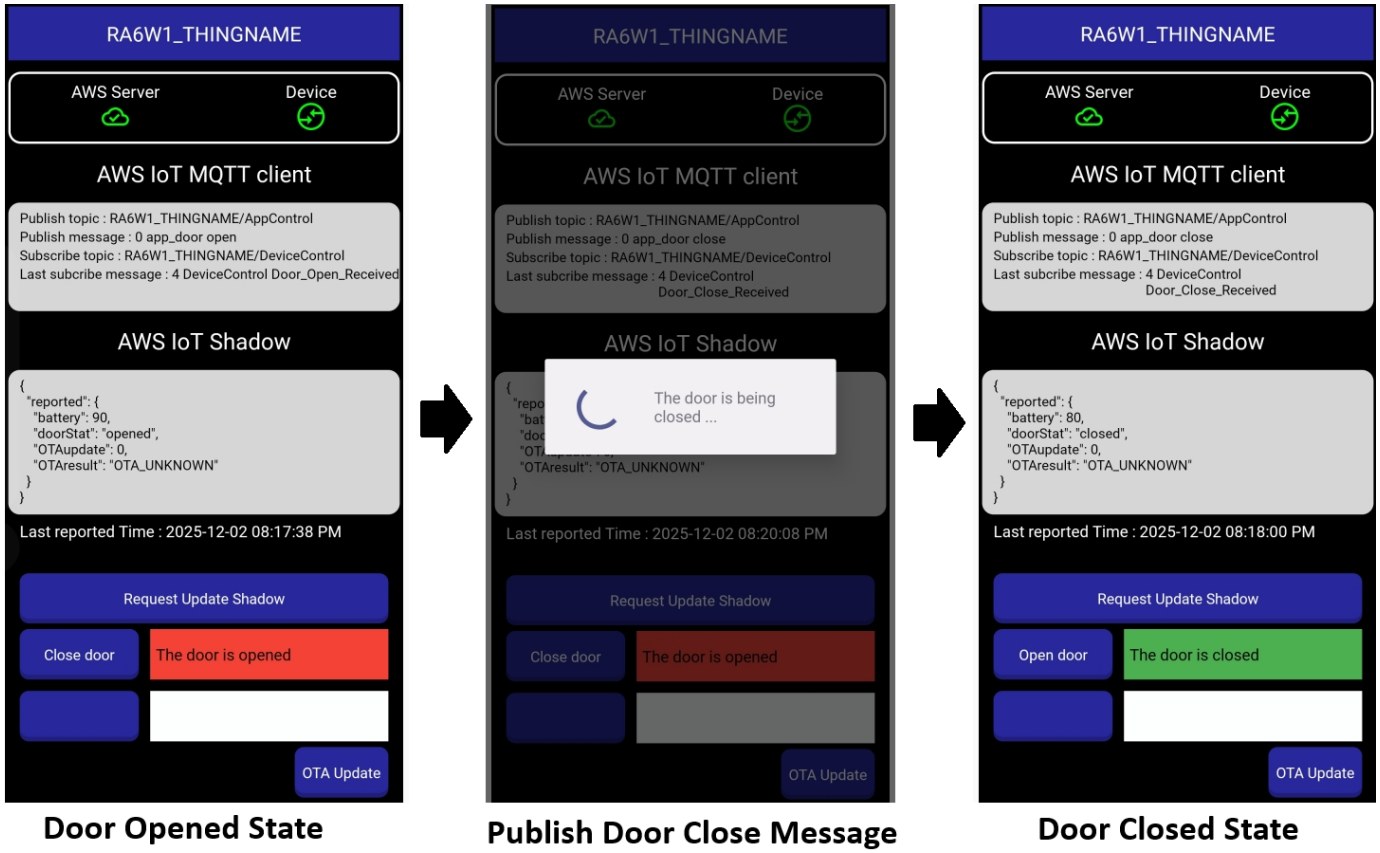

Figure 16. Closing door on mobile app with host MCU

The flow of closing the door is the following, see Figure 16:

1. When you click the **Close door** butto, application publishes message: 0 app_door close on the topic <Thing-name>/AppControl.
2. If the DPM is enabled, when RA6W1 receives the message, it sends an AT command +INIT:WAKEUP,UC. MCU should now keep the DPM from going to sleep by sending the command, AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,4. Now RA6W1 sends the command +AWSIOT=SERVER_DATA 0 app_door close and waits for the response until timeout.
3. If the DPM is not enabled, then RA6W1 sends the command +AWSIOT=SERVER_DATA 0 app_door close and waits for the response until timeout.
4. MCU should respond with the command: AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, as the application is subscribing to the topic <Thing-name>/DeviceControl.
5. MCU should now process the command and do the door opening. RA6W1 now again sends a command +AWSIOT=CMD_TO_MCU update to get the status from MCU and waits for the response until timeout.
6. Now MCU updates the shadow with the command AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80.
7. RA6W1 parses this and publishes the shadow to the application and the box under AWS IoT Shadow is updated according to the received values, also the state of door is changed in the application along with the shadow, see Figure 17.

Console log for closing the door is the following:

```
pal_app_event_cb 0 app_door close
dynamic subscription command ("app_door")
send "SERVER_DATA 0 app_door close" to MCU
=======================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 4 DeviceControl Door_Close_Received
=======================================================
topicCount : 1
Count : 0, cmdNum = 4
mqtttype = 0
data type(publish)=1
call publish: 4 DeviceControl Door_Close_Received
publish (contents update) OK - topic : RA6W1_THINGNAME/DeviceControl payload: 4 DeviceControl Door_
Close_Received
release response
*******************************************************************************
send "CMD_TO_MCU update" to MCU
=======================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 2 doorStat closed 3 battery 80
=======================================================
topicCount : 2
Count : 0, cmdNum = 2
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = closed
Count : 1, cmdNum = 3
mqtttype = 1
index(=0) matched
data type(shadow) = 2
call update sensor(need to be set variable): battery = 80.000000
release response
*******************************************************************************
publish (shadow sensor update) OK -  payload: "{"state":{"reported":
{"battery":80.000000,"doorStat":"closed","OTAupdate":0,"OTAresult":"OTA_UNKNOWN"}},"cli-
entToken":"RA6W1_THINGNAME-0"}"
last user Timer ID = 0
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url: ""
```

AT console log for closing the door is the following:

```
+AWSIOT=SERVER_DATA 0 app_door close
>>>>> command: <AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Close_Received> is given here <<<<<
OK
+AWSIOT=CMD_TO_MCU update
>>>>> command: <AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80> is given here <<<<<
OK
```

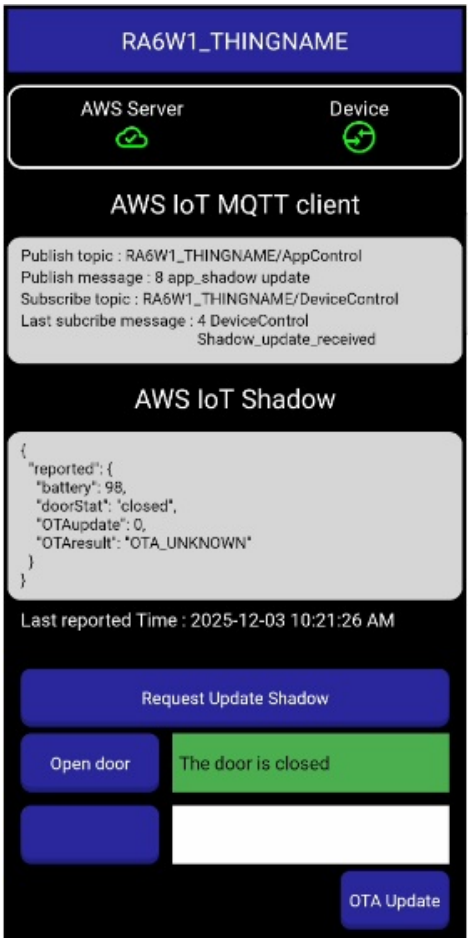

Figure 17. Shadow update on mobile app

The flow of shadow update process is the following, see Figure 17:

1. When you click **Request Update Shadow**, application publishes message: 8 app_shadow update on the topic thingName/AppControl.
2. If the DPM is enabled, when RA6W1 receives the message, it sends an AT command, +INIT:WAKEUP,UC. MCU should now keep the DPM from going to sleep by sending the command AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,1. Now RA6W1 sends the command +AWSIOT=SERVER_DATA 8 app_shadow update and waits for response until timeout.
3. If the DPM is not enabled, then RA6W1 sends the command, +AWSIOT=SERVER_DATA 8 app_shadow update and waist for response until timeout.
4. MCU should respond with the command: AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, as the application is subscribing to the topic <Thingname>/DeviceControl.
5. MCU should now process the command and do the door opening.RA6W1 now again sends a command +AWSIOT-T=CMD_TO_MCU update to get the status from MCU and waits for the response until timeout.
6. Now MCU updates the shadow with the command, AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80.
7. RA6W1 parses this and publishes the shadow to the application and the box under AWS IoT Shadow is updated according to the received values. If the doorStat is changed, the state of open door/close door also changes.

See the following `ttl` script for handling door open, close, and shadow update:

```
; Handles door open/close and resends last command on shadow update
; Initialize variable
strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 98'
doorLockCmdRcvd = 0
:loop
wait '+AWSIOT=SERVER_DATA 0 app_door close' '+AWSIOT=SERVER_DATA 0 app_door open' '+AWSIOT=SERVER_
DATA 8 app_shadow update' '+AWSIOT=CMD_TO_MCU update' '+INIT:WAKEUP,UC' '+INIT:WAKEUP,DEAUTH'
'+INIT:WAKEUP,NOACK'
if result = 1 then
    ; Door closed
    strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80'
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Close_Received'
endif
if result = 2 then
    ; Door opened
    strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat opened 3 battery 90'
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Open_Received'
endif
if result = 3 then
    ; Resend last known command
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Shadow_update_received'
endif
if result = 4 then
    ; Resend last known command
    sendln strlastCommand
    waitln 'OK'
    if doorLockCmdRcvd = 1 then
        sendln 'AT+PMGRCONSTRAINT=2,4'
        waitln 'OK'
        doorLockCmdRcvd = 0
    endif
endif
if (result = 5) || (result = 6) then
    ; Add constraint and notify MCU ready for AT
    sendln 'AT+PMGRCONSTRAINT=1,4'
    ;waitln 'OK'
    sendln 'AT+PMGRMCUWUDONE'
    waitln 'OK'
    sendln strlastCommand
    waitln 'OK'
endif
if result = 7 then
    sendln 'AT+PMGRMCUWUDONE'
    waitln 'OK'
    sendln strlastCommand
    waitln 'OK'
endif
goto loop
```

The `ttl` script can be used as a reference to how the external MCU application should handle the AT commands for door open, close, and shadow update actions.

7.5 AT Command List

7.5.1 Basic Set

Table 1. Basic set of AT commands from MCU to RA6W1

Head	Main	Sub	Parameters
AT+AWS=	SET	APP_ THINGNAME	Set the device thing name. Used to choose a device by its thing name during provisioning.
		AWS_BROKER	Set the broker address.
		APP_LPORT	Set the local port. Default is 443.
		APP_SUBTOPIC	Set subscriber topic name, and the default is "AppControl".
		APP_PUBTOPIC	Set sub-topic name, and the default is "DeviceControl".
		USE_DPM	Define the operation of Sleep mode 3. 0 – DPM Disabled. 1 - DPM Enabled
		SLEEP_MODE	Set Sleep mode. This command is not used currently.
		RTC_TIME	This command is not implemented. It can be used to set the wake-up time for not connected sleep. Where RA6W1 waken up by RTC.
		DPM_KEEP_ ALIVE	This command is not implemented. It can be used to set the keep-alive time between the IoT device and the AP.
		USE_WAKE_UP	This command is not implemented. It can be used to set the wake-up time for Full-boot mode.
		TIM_WAKE_UP	This command is not implemented. It can be used to set the period to check a beacon frame from the AP.
		AWS_USE_FP	This command is not implemented. It can be used to enable/disable fleet provisioning when fleet provisioning feature is implemented.
		APP_BRD_ FEATURE	This command is not implemented. It can be used for board features and Pin MUX.
		UART_CFG	This command is not implemented. It can be used to configure the UART for AT commands.
		APP_MCU_WU_ PORT	This command is not used currently. Set RA6W1 port to wake up external MCU.
		APP_MCU_WU_ PIN	This command is not used currently. Set RA6W1 pin to wake up external MCU.
Example: AT+AWS=SET APP_THINGNAME <Assigned Thing Name> AT+AWS=SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com			

7.5.2 TLS Certificate

Table 2. Write TLS certificate from MCU to RA6W1

Start code	Sub code	Type	End code
0x1B (Esc character)	C0,	Root CA Self-Signed, well known Has root certificate public key Signed by root certificate private key	0x03 (End of Text or Ctrl+C)
	C1	Certificate key Has own public key Signed by root certificate private key Use root certificate public key to prove authenticity	
	C2	Private key Has own public key Signed by certificate private key Use certificate 1 public key to prove authenticity	
Example: send "0x1B" over UART or press Esc. send "C0,-----BEGIN CERTIFICATE-----\n" " MIIDQTCCAimgAwIB ….. -----END CERTIFICATE-----" over UART or copy paste the certificate after C0, send "0x03" or press Ctrl+C			

7.5.3 Configure Attributes

Table 3. Configuration data from MCU to RA6W1

Head	Main	Sub	Parameters
AT+AWS=	CFG	[number] [name] [value-type] [MQTT-type]	■ number: index to identify the saved attribute Increase by 1 when setting a new attribute Max value is 10 (total supported attributes is 10) ■ name: String specifying the attribute name ■ value-type 0 - Integer type 1 - String type 2 - Float type ■ MQTT-type 0 - Publish: The prompt command is used to send a value from the MCU to the phone. For example, door state = true/false: • Shadow: The value is sent to the device twin and updated on the phone the next time it is connected. • Subscribe: The prompt command is used to send a value from the phone to the MCU. For example, door open command.
Example: AT+AWS=CFG 0 app_door 1 2 AT+AWS=CFG 1 app_shadow 1 2 AT+AWS=CFG 2 doorStat 1 1			

7.5.4 Command of MCU to RA6W1

Table 4. Command of MCU to RA6W1

Head	Main	Sub	Description
AT+AWS=	CMD	MCU_DATA	Used by the MCU to set a CFG parameter in the RA6W1.The value must be the same format as defined by the CFG setting. Parameters: [number] [name] [value]
		RESTART	Reboot the device keeping the current mode and status.
		GET_STATUS	Get the current AWS IoT status The MCU can read the current status fromRA6W1 at any time.
		RESET_TO_AP	Not implemented currently.
		FACTORY_RESET	Not implemented currently.
Example: AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80 AT+AWS=CMD RESTART			

7.5.5 Command of RA6W1 to MCU

Table 5. Command of RA6W1 to MCU

Head	Main	Parameters	Description
+AWSIOT	SERVER_DATA	[number] [name] [value]	Used by RA6W1 to set a CFG parameter in the MCU. The value must be the same format as defined by the CFG setting.
	CMD_TO_MCU	update	Used by RA6W1 to request the status of devices such as sensors, batteries, and doors from the MCU. RA6W1 maintains the values obtained from the MCU and forwards them when requested by an external phone app or by an MQTT wake-up event.
Example: +AWSIOT SERVER_DATA 0 door_control open+AWSIOT CMD_TO_MCU update			

7.5.6 Status from RA6W1 to MCU

Table 6. Status from RA6W1 to MCU

Status	Value	Parameters
IDLE	-1	Initial state of AWS-IoT application. Sent when a system error occurs. For example, network connection failure
Done factory reset	0	Sent after completes factory reset
Boot Ready	1	Sent when entering AWS-IoT application mode
Need configuration	5	Sent if there is no setting. MCU should set and configure with the SET and CFG command
Start AP mode	10	Sent when being started to AP mode
Network OK	15	Sent when it is OK to connect AP without problem
Network fail	16	Sent when it fails to connect AP with any problem
Start STA	20	Sent when being started to STA mode
Done STA	25	Sent when entering Sleep mode for DPM
MCUOTA	30	Sent when MCU OTA starts processing
Example: +AWSIOT STATUS 15		

8. AWS IoT

The RA6W1 is an SoC for IoT applications such as security systems, door locks, and smart applications. This section provides procedures on how to configure AWS IoT for communicating with the RA6W1 IoT device.

To connect a device to the AWS IoT server, the following components are required:

1. Sign up AWS account and permissions.
2. Connect devices to AWS IoT.

To create an AWS account and grant permissions:

1. Go to AWS website and create a free account (https://portal.aws.amazon.com/).
2. Create an administrative user for performing daily administrative tasks.
3. Open the AWS IoT console to get started with AWS IoT.

NOTE
While creating the project, certificates and policies, the screenshots may look slightly different from what is shown in the document, and you need to use navigation in the AWS IoT core environment to find corresponding attributes.

8.1 Connect Devices to AWS IoT

You can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of IoT solutions through AWS IoT console. Prior to sending data to and receiving data from AWS IoT server, you should register a device first.

In the **Thing Registry**, the devices connected to the AWS IoT server are **Things**. The **Thing Registry** allows keeping records of all devices that are connected to an AWS IoT account.

To register a device in the Thing Registry:

1. On the AWS IoT console, expand **Registry**.
2. Expand **All devices** and click **Things** > **Create things**, see Figure 18.

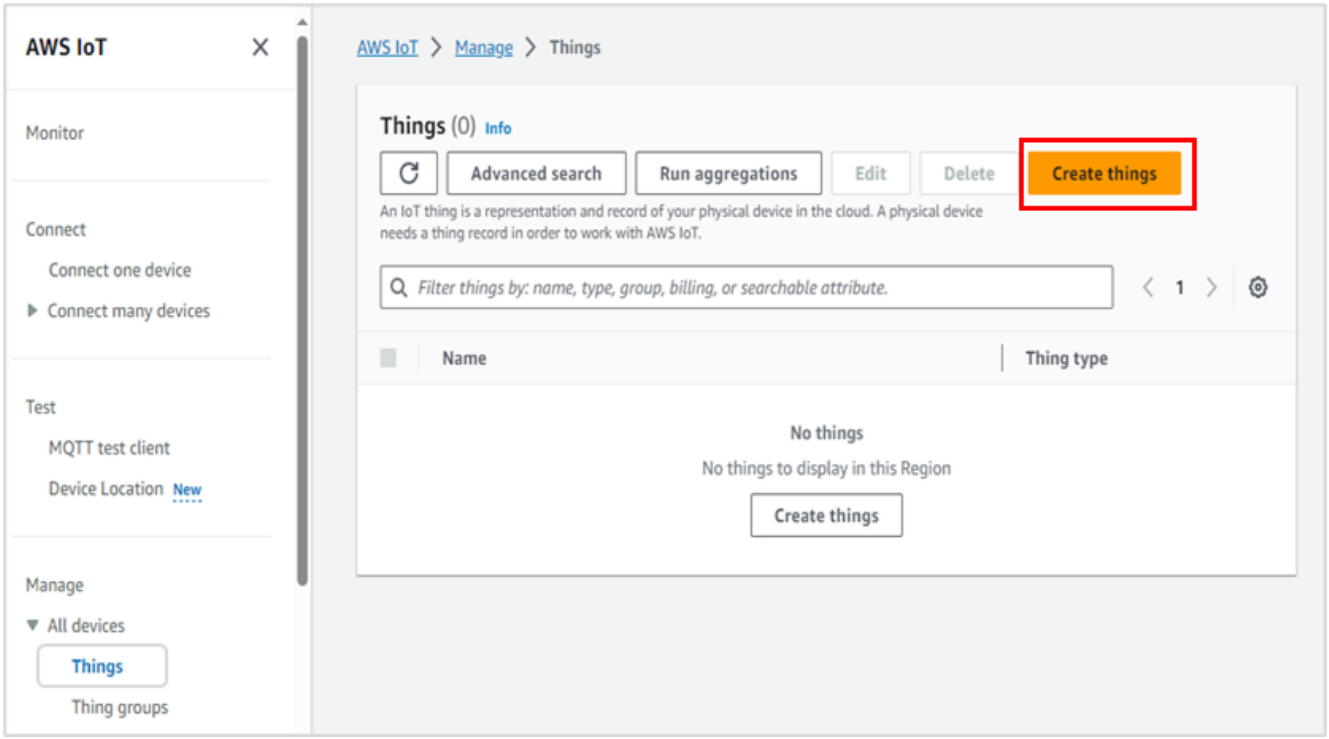

Figure 18. Create things

3. Select **Create single thing**.

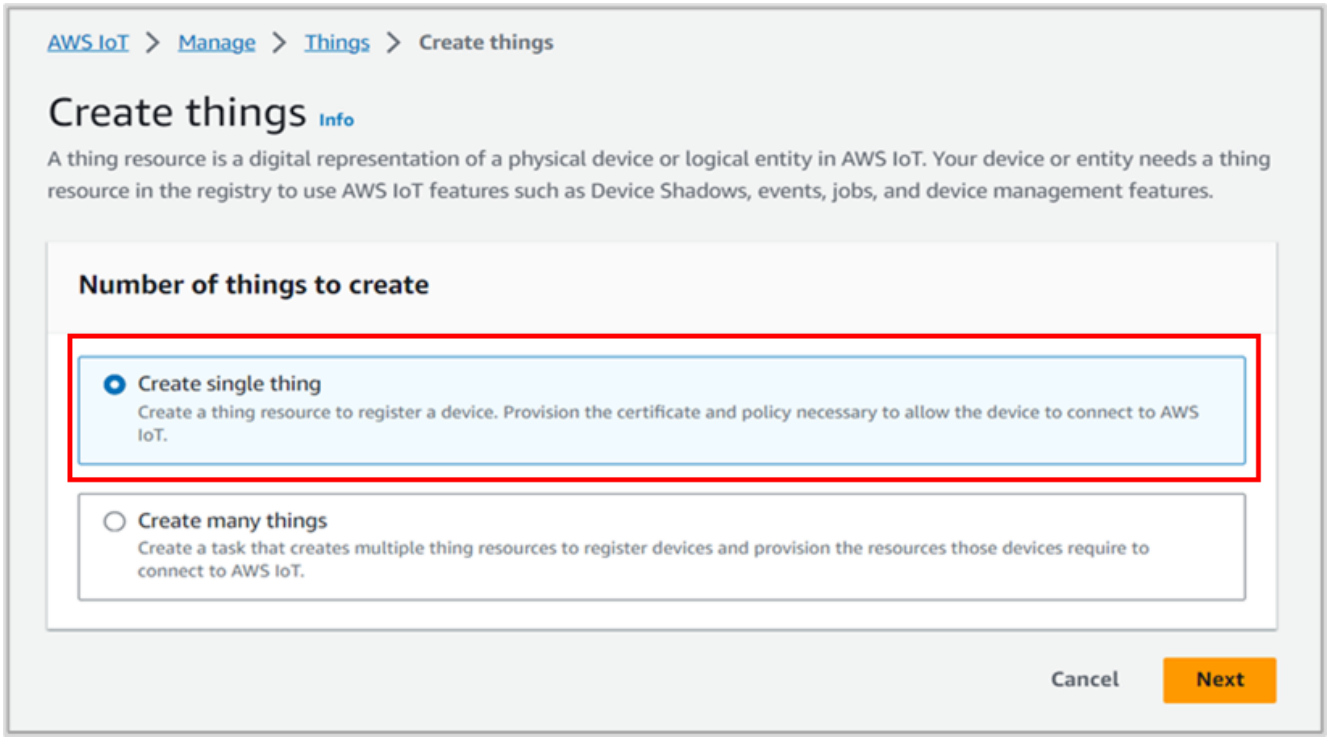

Figure 19. Create single thing

4. Specify the needed thing properties, in the **Thing name** field, enter a device name, for example, "MyTestDoorLock", and on the **Device Shadow**, select **Unnamed shadow (classic)** and click **Next**, see Figure 20.

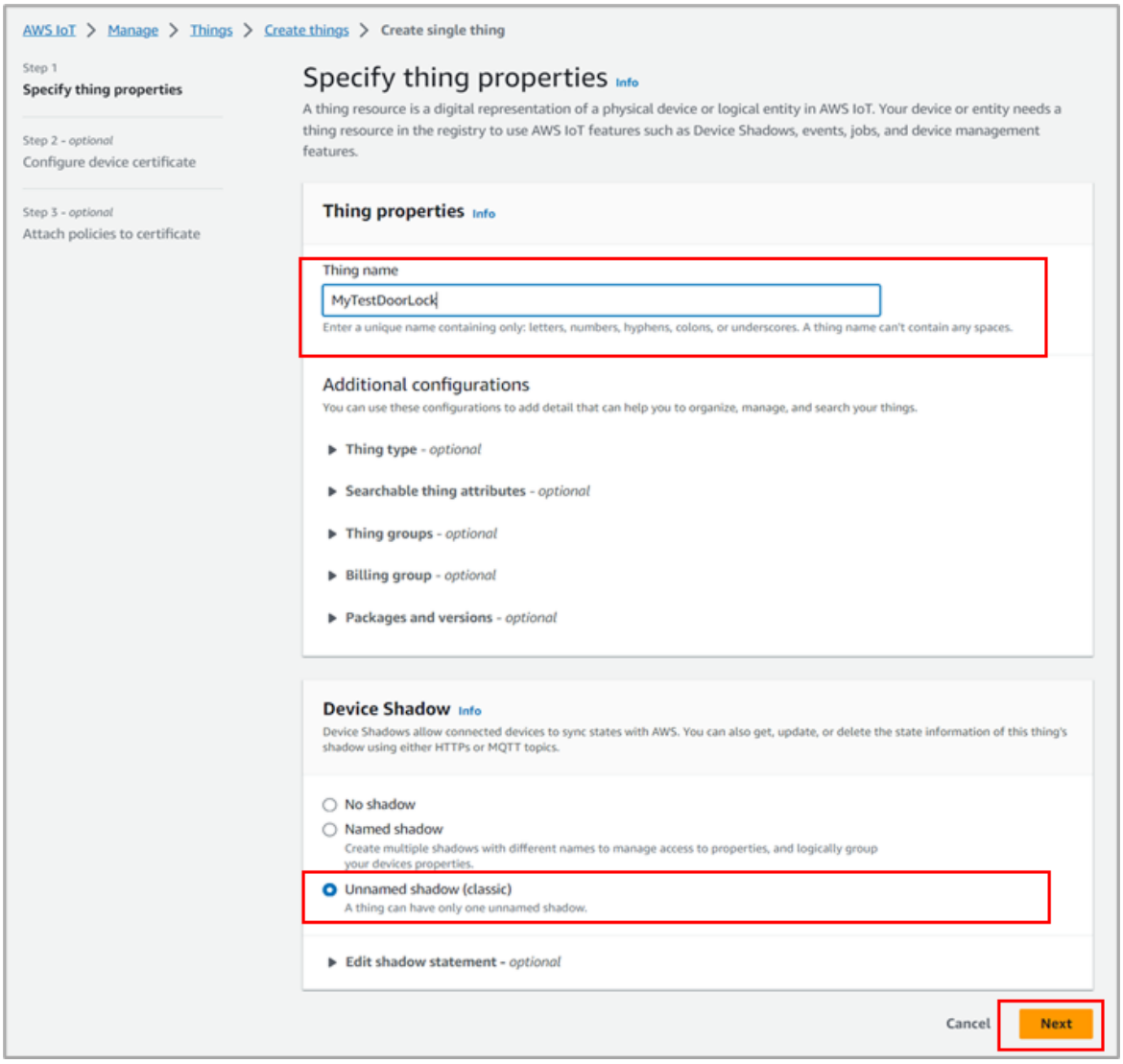

Figure 20. Thing properties

5. Select **Skip creating a certificate at this time** and click **Create thing**.

   Now you have the thing created to perform the test and it is named **MyTestDoorLock**.
6. In the **Things**, select the created thing and the thing details appear.
7. For the shadow function of the thing, select the **Device Shadows** and select **Classic shadow**.

Figure 21 shows the device shadow document.

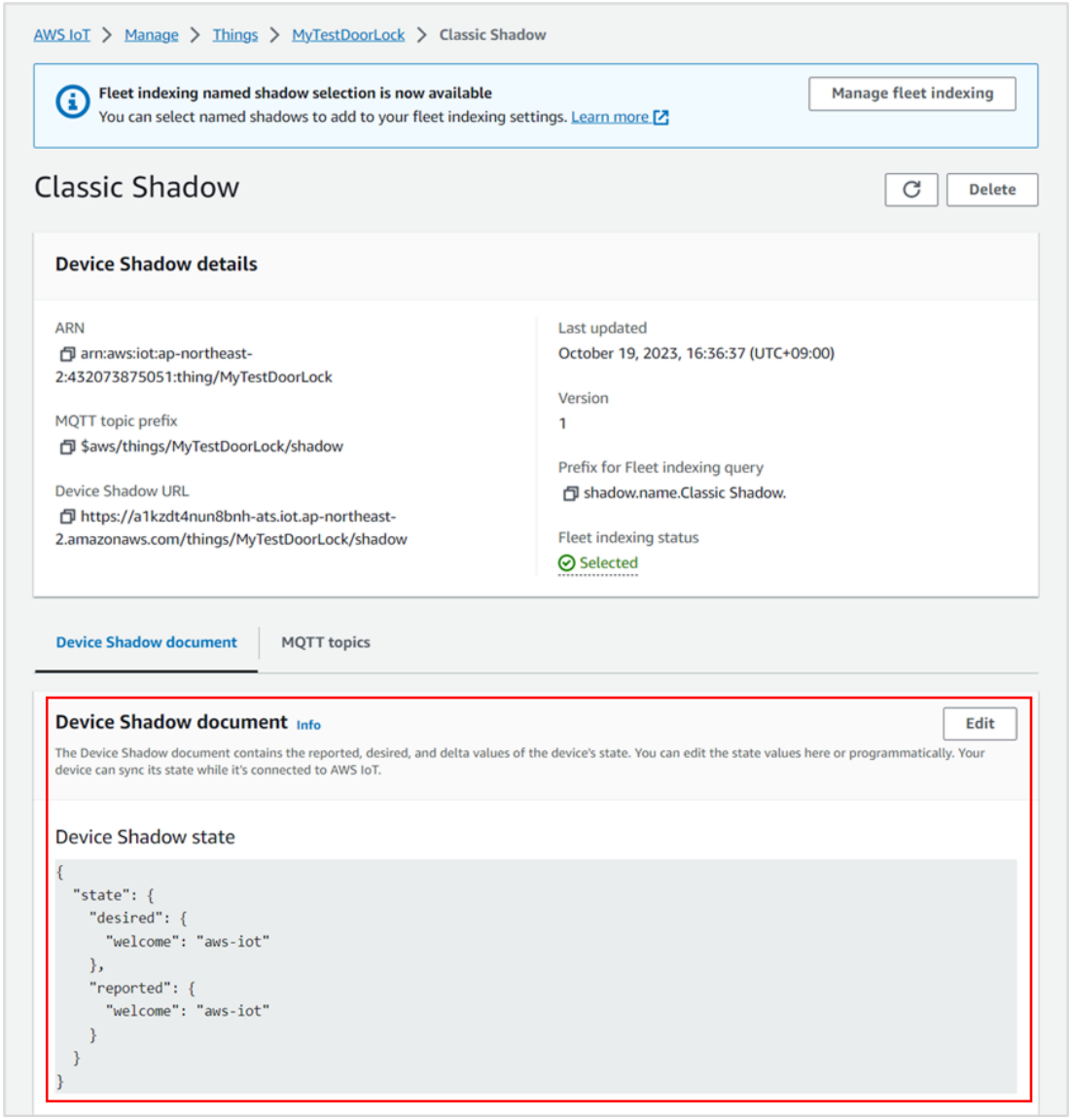

Figure 21. Device shadow document

For more information on device shadows for AWS IoT, visit AWS IoT Device Shadow service (https://-docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html).

8.1.1 Create and Activate Device Certificate

The communication between the device and the AWS IoT web service is protected by X.509 certificates. You can let the AWS IoT generate a certificate or you can use your own X.509 certificate. This section shows that AWS IoT generates the X.509 certificate.

You should activate the certificates before using it. To create and activate a device certificate:

1. On the navigation pane, expand **Security** and click **Certificates**, and then click **Add certificate** > **Create certificate**, see Figure 22.

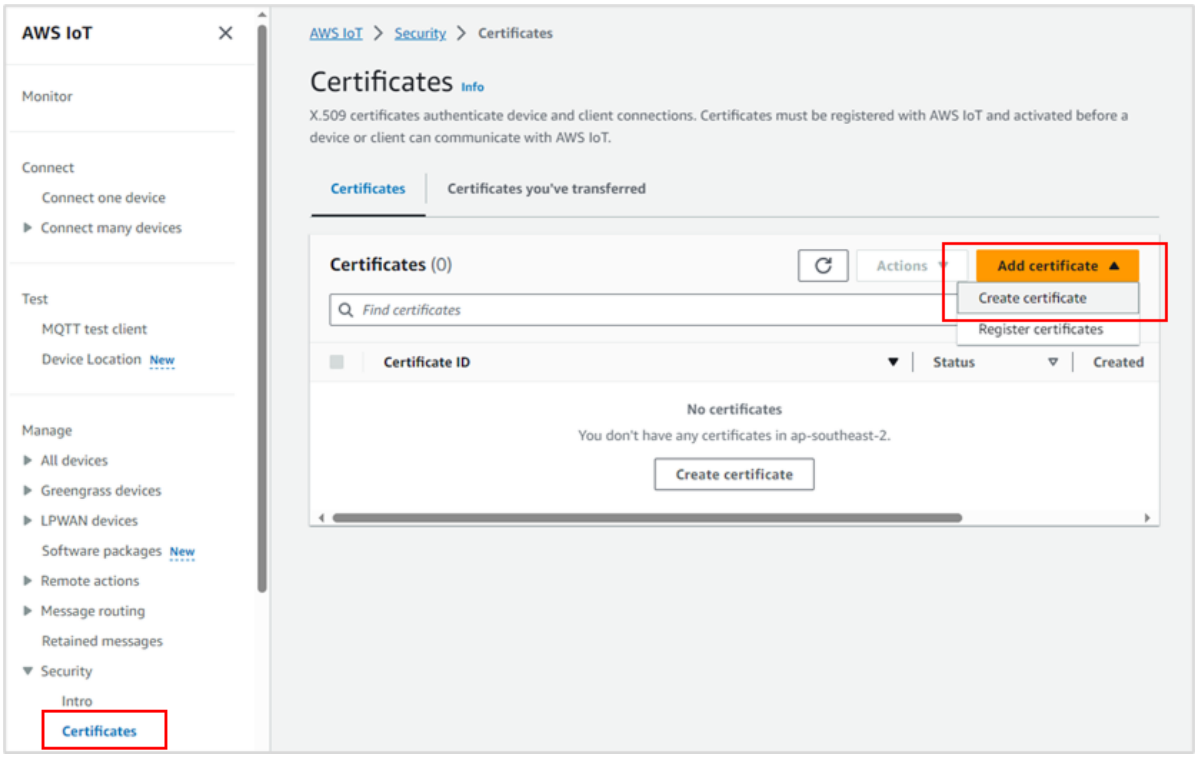

Figure 22. Create certificates

2. Select **Auto-generate new certificate (recommended)** and **Active** options, and then click **Create**.

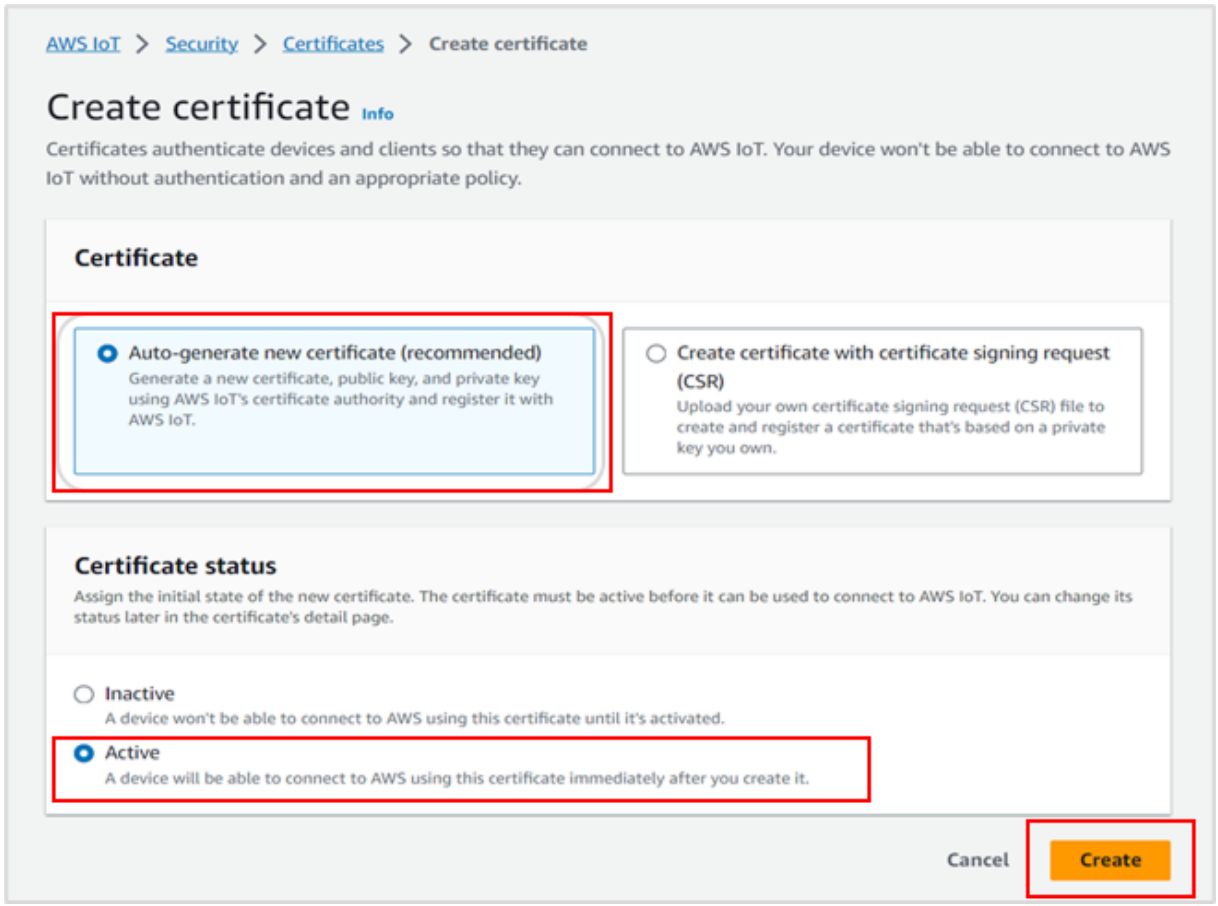

Figure 23. Create certificates (continued)

There are three required certificates to download.
On the **Certificate Created** page, to download the device certificate, private key, and root CA certificates for AWS IoT, and save the downloads to your computer, click **Download**.

NOTE
You must save the certificate files before leaving this page. After leaving this page in the console, you no longer have access to the certificate files. Renesas recommends that Device certificate, Private key file, and Root CA should be downloaded in sequential order.

For Root CA, visit the AWS Documentation site (https://docs.aws.amazon.com/iot/latest/developerguide/server-authentication.html#server-authentication-certs). Root CA certificates are subjected to expiration and/or revocation.

The certificate status should be **Active** in the list of certificates, see Figure 24.

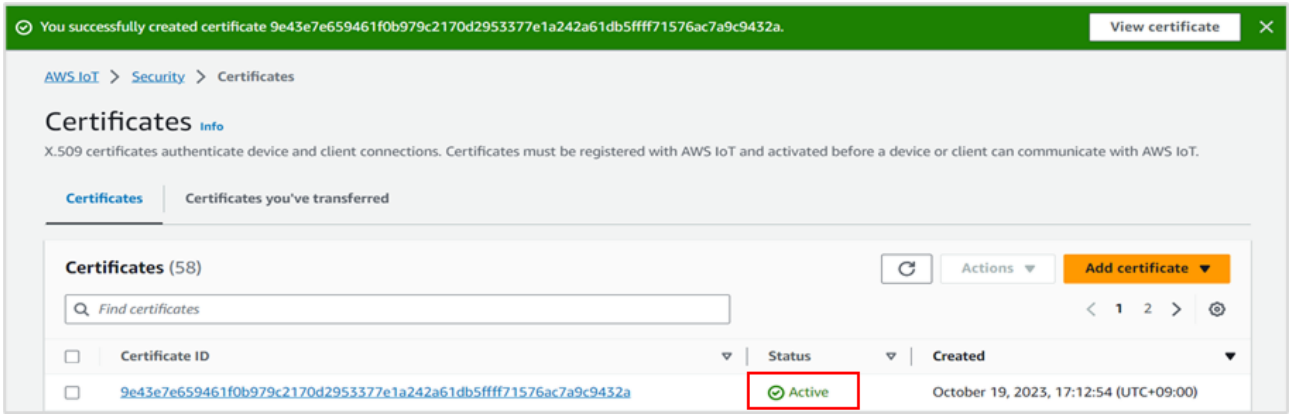

Figure 24. Active certificate

The X.509 certificates are used to authenticate the device with the AWS IoT. The AWS IoT policies are used to authorize the device for AWS IoT operations, such as subscribing or publishing to MQTT topics. The device displays its certificate only while connecting to the AWS IoT.

To allow the device for AWS IoT operations, you should create an AWS IoT policy and attach that policy to the device certificate.

To create an AWS IoT policy:

1. On the navigation pane, expand **Security** > **Policies** and click **Create policy**

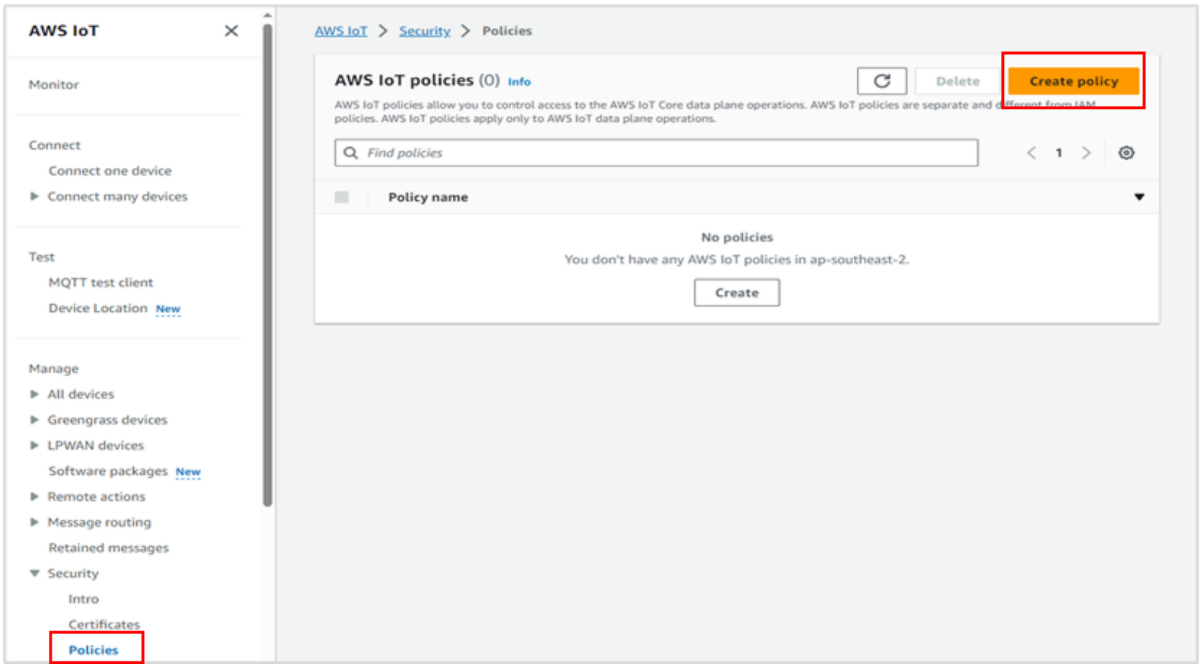

Figure 25. Create policy

2. On the **Create policy** page:

   a. In the **Policy name** field, under **Policy properties**, enter a name for the policy (for example, MyTestPolicy). Renesas Electronics recommends not using personally identifiable information in policy names, see Figure 26.

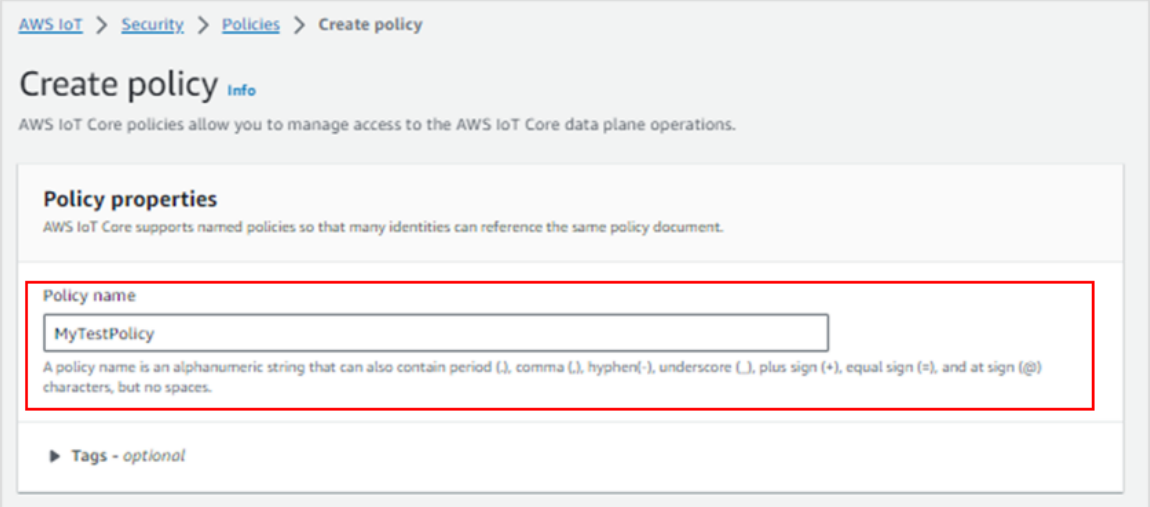

Figure 26. Add policy name

b. Set **Action** for the policy to **iot:***
c. Set **Resource ARN** to *
d. After entering the required information, click **Create**.

NOTE
The examples in this document are intended only for development environments. All devices in your production fleet must have credentials with privileges that authorize only intended actions on specific resources. The specific permission policies may vary depending on use cases. Identify the permission policies that best meet the business and security requirements. For more information, see Example Policies and Security Best practices in AWS IoT.

3. To view the created policies, expand **Security** and click **Policies**, and select the created policy.

After an AWS IoT policy is created, you must attach that policy to the device certificate. The attachment of an AWS IoT policy to a certificate gives the device the permissions that are specified in the policy.

To attach the AWS IoT Policy to a device certificate:

1. Go to the certificate created by you, select **Policies**, and click **Attach policies**.
2. Select the created policy and click **Attach policies**.

NOTE
A device should have a certificate, private key, and root CA certificate to authenticate with the AWS IoT. Renesas recommends that you attach the device certificate to the thing that represents the device in AWS IoT. This allows you to create AWS IoT policies that grant permissions based on certificates attached to things.

3. Go to the certificate created by you, select **Things**.
4. Selected created thing and click **Attach to things**.
5. To activate the certificate, click **Activate** from the **Actions**.

8.2 AWS Core APIs

The AWS Core MQTT is documented online. Table 7 lists APIs provided by AWS Core MQTT that are used as a part of the Application Example.

Table 7. MQTT module APIs

API	Description
MQTT_Init	Initializes an MQTT context.
MQTT_Connect	Establishes an MQTT session.
MQTT_Subscribe	Sends MQTT SUBSCRIBE for the given list of topic filters to the broker.
MQTT_Publish	Publishes a message to the given topic name.
MQTT_Ping	Sends an MQTT PINGREQ to broker.
MQTT_Unsubscribe	Sends MQTT UNSUBSCRIBE for the given list of topic filters to the broker.
MQTT_Disconnect	Disconnect an MQTT session.
MQTT_ProcessLoop	Loop to receive packets from the transport interface. Handles keep-alive.
MQTT_ReceiveLoop	Loop to receive packets from the transport interface. Does not handle keep-alive.
MQTT_GetSubAck-StatusCodes	Parses the payload of an MQTT SUBACK packet that contains status codes corresponding to topic filter subscription requests from the original subscribe packet..
MQTT_Status_strerror	Error code to string conversion for MQTT statuses.
MQTT_PublishToResend	Gets the packet ID of next pending publish to be resent.

8.3 FSP APIs

8.3.1 AWS IoT Port Layer (rm_awsiot_w)

The only two APIs: RM_AWSIOT_W_Open and RM_AWSIOT_W_Close in FSP module rm_awsiot_w are not implemented currently. So, they are not used in the reference application.

8.3.2 AWS LWIP Sockets Wrapper (rm_aws_lwip_sock_wrap_w)

This section lists the APIs for the rm_aws_lwip_sock_wrap_w (Networking) Module. The module provides interface to control, access, write to lwIP. The APIs listed in the Table 8 is used to manage the DPM for AWS functionality.

Table 8. AWS LWIP sockets wrapper FSP APIs

API	Description
app_dpm_get_sleep_flag	Retrieves the flag indicating whether the API enters or exist DPM.
app_dpm_set_sleep_flag	Set the flag indicating whether the system should enter or exit DPM.
app_dpm_info_get	Get if exist or allocate if not exist the RTM resource for server apps.
app_dpm_get_client_socket_port	Get the saved client binding port.
app_dpm_set_send_pub_flag	Set the flag whether device publish to server or not.
app_dpm_get_send_pub_flag	Get the flag whether device publish to server or not.
app_dpm_set_wait_job_next_flag	Set the flag whether device need to wait the flag for next job or not.
app_dpm_get_wait_job_next_flag	Get the flag whether device need to wait the flag for next job or not.
app_get_peer_ip_str	Get the IP string of server to connect.
app_get_random_local_port	Retrieves a randomly generated local port number for socket binding.
app_get_count_of_reconnection	Get the number of reconnection attempts on DPM Wake-up mode.
app_socket_set_port_n_filter	Register a binded local port for DPM mode and set port filter for DPM wake-up.
app_set_persistent_session	Set the flag depending on whether the persistent session exists or not.
app_is_persistent_session	Retrieves the flag indicating whether a persistent session is enabled.
app_is_reconnected	Get the flag whether reconnection happened or not..
app_set_reconnect_flag	Set the reconnection flag to the input parameter.
app_dpm_get_recv_timeout_flag	Get the receive timeout flag.
app_dpm_set_recv_timeout_flag	Set the receive timeout flag to the input parameter.
app_dpm_get_unknown_uc_flag	Get the unknown DPM wake-up cause flag.
app_dpm_set_unknown_uc_flag	Set the unknown DPM wakeup cause flag to the input parameter.
app_set_ka_value	Set the keepalive time interval for connection peer to the input parmeter by seconds.
app_get_ka_value	Get the keepalive time interval for communication with peer.
app_dpm_set_rcv_ready	Notify the DPM module that the receiver is ready after DPM wake-up.
persistant_read_callback_set	Set read callback function to get server IP Address from nvram.
persistant_write_callback_set	Set write callback function to set server IP Address to nvram.
awsiot_app_print_elapse_time_ms	Check passed time.

The reference applications, rm_awsiot_w_aws_example_ek_ra6w2 and rm_awsiot_w_aws_atcmd_example_ek_ra6w1 shows how to implement the APIs for AWS IoT applications.

See the FSP documentation which can be downloaded from Renesas website for more explanation of the APIs.

9. Revision History

Revision	Date	Description
1.01	Dec 18, 2025	■ Modified Section 6 Build and Run Reference Application ■ Added Section 7 Reference Application using AT Commands. ■ Added Section 8.3 FSP APIs
1.00	Aug 31, 2025	Initial version.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.1 Jan 2024)

CorporateHeadquarters

TOYOSU FORESIA, 3-2-24 Toyosu
Koto-ku, Tokyo 135-0061, Japan

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-us/