

AN12199

A71CH for secure connection to Google Cloud IoT Core

Rev. 1.1 — 12 September 2018
485611

Application note
COMPANY PUBLIC

Document information

Info	Content
Keywords	Security IC, IoT, Google Cloud IoT Core, authentication, ECC
Abstract	This application note describes how to set up a trusted connection to Google Cloud IoT core using the A71CH Customer Programmable type.



Revision history

Rev	Date	Description
1.1	20180912	Addition of Section 5
1.0	20180608	First document version

Contact information

For more information, please visit: <http://www.nxp.com>

1. Introduction

One critical component in IoT is device identity. The cloud platform needs to be able to verify the identity of the device to trust its data and grant access to the cloud platform. As such, the IoT device identity should be unique, verifiable, and trustworthy to establish a root of trust. The IoT device identity can be verified using public key cryptography. Public key cryptography algorithms are based on key pairs: one private key and one public key.

The private key must remain secret for the entire lifetime of the product to avoid malicious attackers, who can falsify the identity of your devices. The private key must be isolated from users, software, and flash memory of microcontrollers to achieve confidentiality. The leakage of any of the private keys involved in the root of trust leads to serious security threats.

The A71CH can prevent key leakage by providing a tamper-resistant platform, capable of securely storing and provisioning credentials, securely connecting IoT devices to cloud services and performing cryptographic node authentication. The A71CH solution offers an outstanding level of security measures, which protect the IC against physical and logical attacks. In addition, it can be used with various host platforms and host operating systems to secure a broad range of applications.

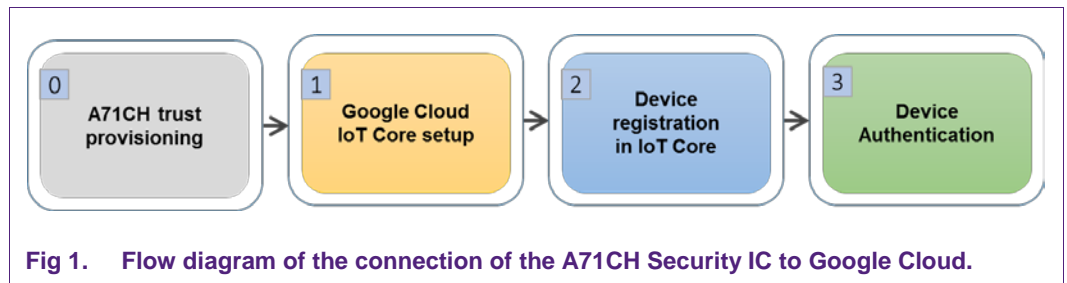
2. Google Cloud IoT Core setup

Google Cloud IoT Core is a fully managed service that allows you to easily and securely connect, manage, and ingest data from millions of globally dispersed devices. Google Cloud IoT Core, in combination with other services on Google Cloud IoT platform, provides a complete solution for collecting, processing, analyzing, and visualizing IoT data in real time to support improved operational efficiency. Google Cloud IoT Core offers the following security features:

- Per-device public/private key authentication using JSON Web Tokens (JWT).
- Signature verification supported by RSA or Elliptic Curve algorithms.
- Support for key rotation per device by allowing concurrent keys to be registered, and support for expiration time per credential.
- TLS 1.2 connection (optional, using root Certificate Authority certificate).
- Cloud IoT Core API access is controlled by Cloud Identity and Access Management (IAM) roles and permissions.

To establish a trusted connection to Google Cloud IoT Core, a *logical device* needs to be created in the platform by registering the *public key* or *device certificate* of the IoT device in the Google Cloud IoT Core *Device Manager*. The corresponding private key is securely stored in the A71CH and used in a confidential manner during the entire product lifecycle. IoT devices are connected by completing the following steps:

1. **Prerequisite:** A71CH provisioned with the required credentials.
2. Set up the Google Cloud IoT Core account.
3. Register IoT device into Google Cloud IoT Core.
4. IoT device authentication



The following sections describe all the steps in this diagram in further detail. Fig 2 shows the connection flow when the device is registered using its public key, while Fig 3 shows the same flow when a device certificate is used for this purpose.

2.1 Google Cloud IoT Core setup

An account and a project need to be created in the Google Cloud IoT Core platform. To do so:

1. Download Google Cloud SDK.
2. Sign in to your Google Account and go to Google Cloud Platform.
3. Choose a project or create a new one.

2.2 IoT device registration in IoT Core

The next step is to register the IoT device credentials. The Google Cloud IoT Core API, `gcloud` commands or the Google Cloud Platform (GCP) console can be used for this purpose. The steps required are:

1. Create a device registry.
2. Create a logical device specifying the credentials created. It will be used to verify the identity of the device.

An IoT device can be registered either by using the IoT device public key or the IoT device certificate.

2.3 IoT device authentication

If the IoT device is successfully registered, it can establish a MQTT trusted connection over TLS. The IoT device authentication consists of the following steps:

1. The IoT device prepares a JWT, which is signed with the private key created stored in the A71CH security IC.
2. The IoT device creates an MQTT Client and connects TLS.
3. Connect MQTT Client to MQTT Broker.
4. The MQTT Broker verifies the JWT signature against the public key of the device.
5. The MQTT Broker accepts the connection.
6. The connection is closed when the JWT expires.

Fig 2 shows the complete IoT device authentication flow:

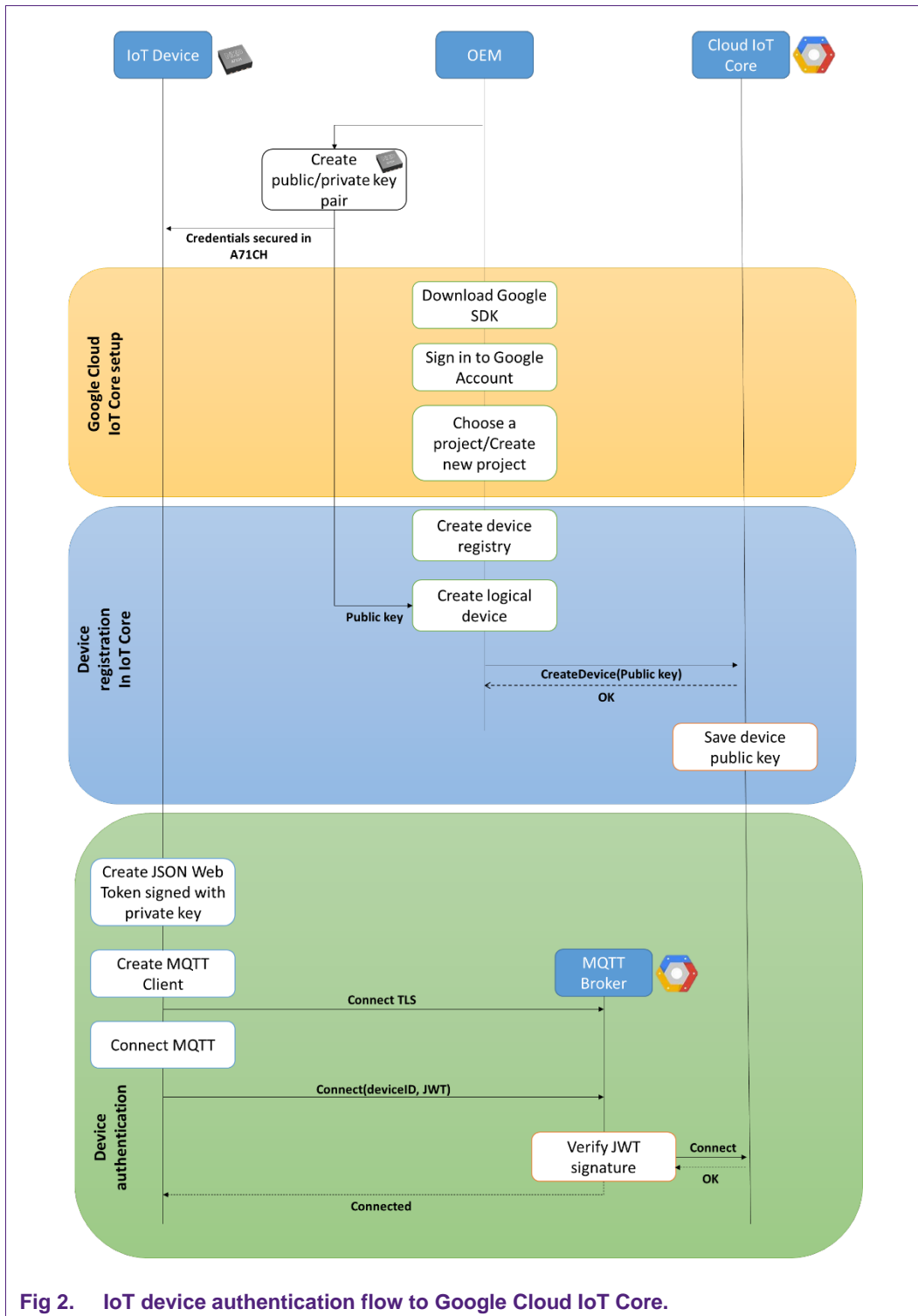
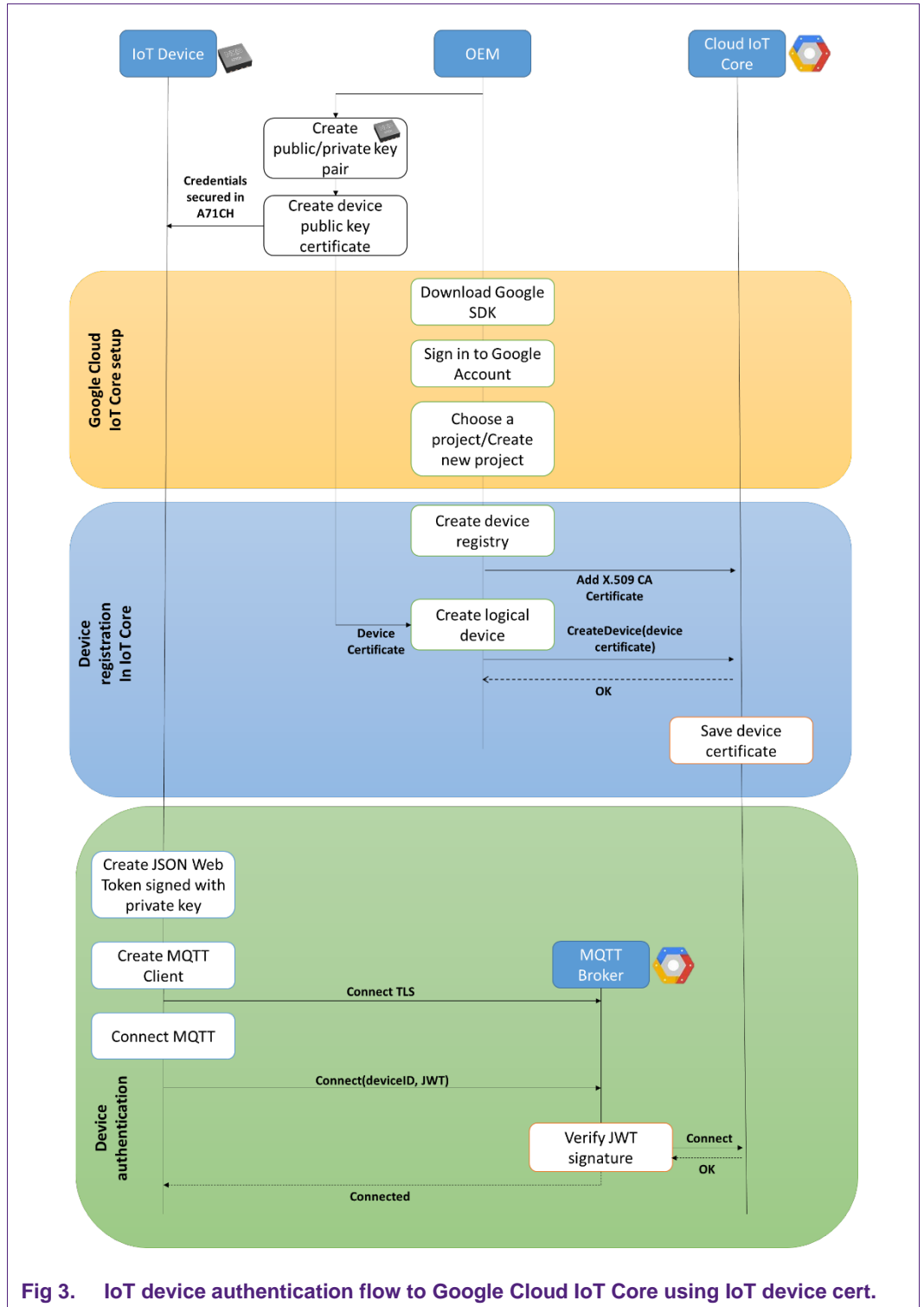


Fig 3 shows the complete IoT device authentication flow when the IoT device certificate is used for this authentication:



3. Trusted connection to Google Cloud IoT Core process

In this chapter, the steps described in the flow diagrams of Fig 2 and Fig 3 are explained in detail when using the Google Cloud Platform Console (GCP) and `gcloud` Command Line Interface (CLI). The corresponding commands from the Cloud API can be found in [GCLOUD_API].

3.1 Creation of credentials

As a prerequisite, it is assumed that the A71CH security ICs are provisioned with their corresponding device credentials. Please refer to Section 5 for the different A71CH Trust Provisioning options.

3.2 Google Cloud IoT Core setup

To work with Google Cloud Platform, and with the IoT Core, it is necessary to download Google Cloud SDK. It is a set of tools that can be used to manage resources and applications hosted on Google Cloud Platform. The `gcloud` command-line tool is downloaded along with the Cloud SDK. It can be downloaded from [CLOUD_SDK].

Once the installation has finished, it is recommended to run the following command to update all the components of the SDK.

```
$ gcloud components update
```

3.2.1 Google Account login and project selection/creation.

The process to setup Google Cloud IoT Core can be performed using either GCP Console (Section 3.2.1.1) or `gcloud` CLI (Section 3.2.1.2).

3.2.1.1 Using GCP Console

Sign in to your Google Cloud IoT Core account. In Fig 4, the GCP Console home page is shown. Using the left menu, the different functions of Google Cloud can be accessed. Click on the '*Select a project*' option, and either choose an existing project or create a new one by choosing '*New Project*'.

To be able to access the Cloud IoT Core API, it is required to enable a billing option. You can define a billing account by choosing the '*Billing*' option on the left menu. The next step is to enable the Cloud IoT Core API, by clicking on the '*APIs & Services*' option on the left menu.

3.3.1.1 Device registry creation using GCP Console

First, go to the 'IoT Core' option in the left menu. Then, click on the 'Create a device registry' option. Write the ID of the registry, choose the region (us-central1, europe-west1, asia-east1), select the MQTT communication protocol and select at least one Telemetry topic. Each topic maps to a specific subfolder in the MQTT topic path when the data is published. If only one Cloud Pub/Sub is selected, all device telemetry will be published on that topic. If multiple topics are defined, you should also include one default topic that has no subfolders. This default topic will be used for published telemetry events that do not have a subfolder or if the subfolder does not have a matching Pub/Sub topic.

It is possible to validate device public key certificates against registry-level Certificate Authority (CA) certificates. In this case, it is necessary to add a CA certificate using the Add Certificate option of the 'Create a device registry' page.

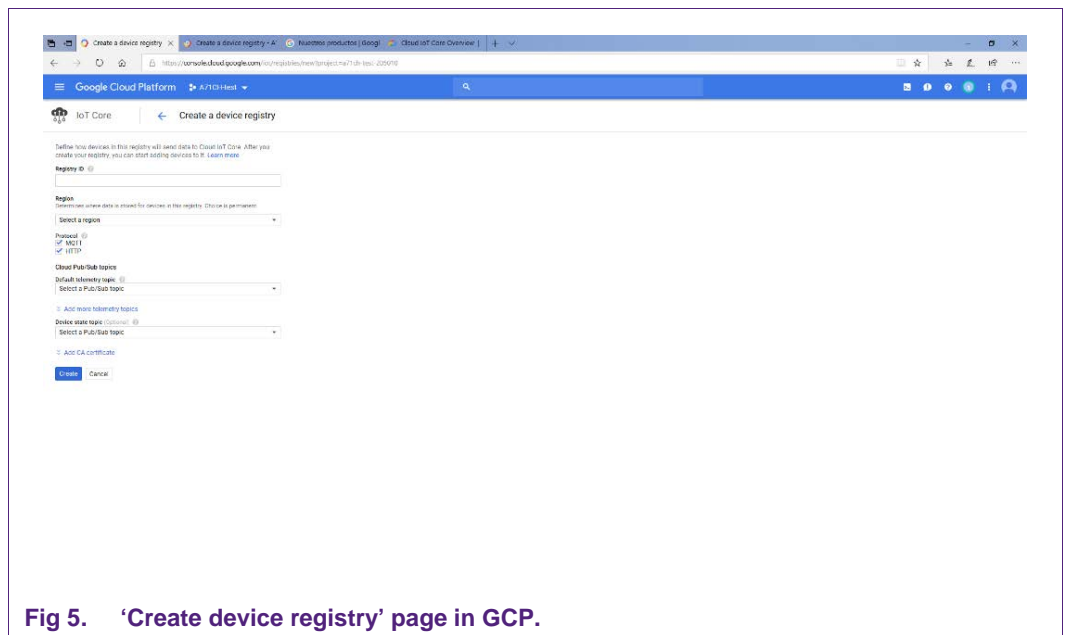


Fig 5. 'Create device registry' page in GCP.

3.3.1.2 Device registry creation using gcloud

To create a registry, run the `gcloud iot registries create` command:

```
$ gcloud iot registries create REGISTRY_ID \
--project=PROJECT_ID \
--region=REGION \
[--event-notification-config=topic=TOPIC] [--state-pubsub-
topic=STATE_PUBSUB_TOPIC]
```

Specify the ID of the registry (`REGISTRY_ID`), the ID of the project where you will create the registry (`PROJECT_ID`) and the region (`REGION`). You can also enable or disable MQTT and HTTP protocols for the registry (both are active by default), by using the `--enable-http-config` and `--enable-mqtt-config` flags. You can specify multiple Pub/Sub topics and their matching subfolders by repeating the `--event-notification-config` flag.

In addition, to add a CA certificate from the `gcloud` CLI, run the following command:

```
$ gcloud iot registries credentials create \
--path=PATH_TO_CERTIFICATE --project=PROJECT_ID \
--registry=REGISTRY_ID --region=REGION
```

3.3.2 Create a logical device

To create a logical device, the public key of the device or, optionally, a device certificate is needed. In the following subsections, the required steps using either the GCP Console or `gcloud` CLI are explained.

3.3.2.1 Logical device creation using GCP Console

Go to the ‘*Device registries*’ page and click the ID of the registry where you want to create the logical device. The ‘*Registry device*’ page opens, then click on ‘*Add device*’. You will need to define a *Device ID* (you will not be able to change it later) and add the public key created in the previous section. You must also select the public key format you are using. In case you are using a device certificate, you need to choose the `ES256_X509` option and upload the certificate.

It is also necessary to select Allow or Block for the device communication. This option allows blocking the communication if necessary; e.g., when the device is not functioning correctly. Make sure that the public key format field matches the key pair for this device.

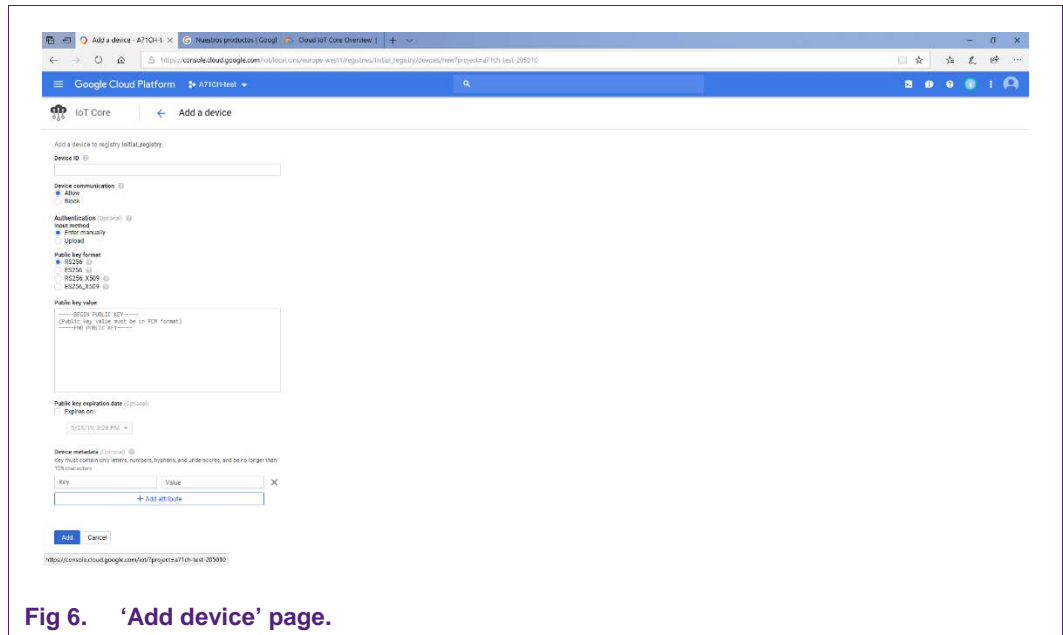


Fig 6. ‘Add device’ page.

3.3.2.2 Logical device creation using `gcloud`

To create a device, run the `gcloud iot device create` command. It is necessary to specify the ID of the device to be created (`DEVICE_ID`), the project and the registry where it will be created (`PROJECT_ID` and `REGISTRY_ID`) and the public key that has been created (`--public-key` flag).

```
$ gcloud iot devices create DEVICE_ID --project=PROJECT_ID \
--region=REGION -registry=REGISTRY_ID \
--public-key [path=PATH, type=TYPE]
```

In case you are using a device certificate, you need to upload the signed public key and specify it setting the `type` flag to `es256-x509-pem`.

3.4 Device authentication

As shown in Fig 2 and Fig 3, to authenticate an IoT device to Google Cloud IoT Core, the device will prepare a JSON Web Token (JWT) signed with the private key of the IoT device. This JWT will be sent to the Cloud IoT Core through an MQTT Bridge running across a TLS connection. Also, it can be observed that a TLS connection between the MQTT Broker and the MQTT Client (IoT Device) is configured.

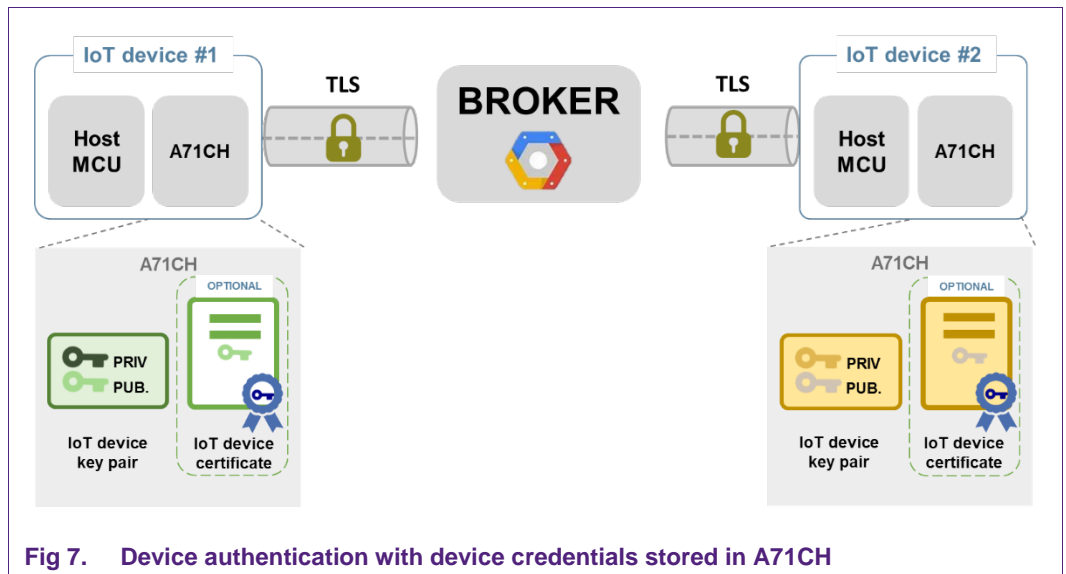


Fig 7. Device authentication with device credentials stored in A71CH

3.4.1 JSON Web Tokens

JSON Web Tokens [JWT_LIBS] provides a secure signing mechanism that verifies and sends information between IoT devices and Google Cloud IoT Core. They are composed of three sections: a *header*, a *payload* and a *signature*. The header and payload are JSON objects that are serialized to UTF-8 bytes, then encoded using Base64url encoding. The signature uses the private key of the IoT device.

```
$ gcloud iot devices create DEVICE_ID --project=PROJECT_ID \
--region=REGION -registry=REGISTRY_ID \
--public-key [path=PATH, type=TYPE]
```

The JWT Header consists of two mandatory fields that indicate the signing algorithm and the type of token. For Elliptic Curve keys, the JSON representation of the header is as follows:

```
{ "alg": "ES256", "typ": "JWT" }
```

The JWT Payload contains a set of claims, which contain information about the Token. The set of required claims are:

- “Issued At” (iat): Timestamp when the token is created, specified as seconds since 00:00:00 UTC, January 1, 1970.
- “Expiration” (exp): Timestamp when the token stops being valid, specified as seconds since 00:00:00 UTC, January 1, 1970.
- “Audience” (aud): Single string containing the PROJECT_ID where the device is registered.

```
{ "aud": "PROJECT_ID", "iat": iat_timestamp, "exp": exp_timestamp }
```

Then, the header and the payload are Base64url-encoded. This Base64url-encoded message is signed using the private key stored in A71CH security IC. The result of the signature is also Base64url-encoded.

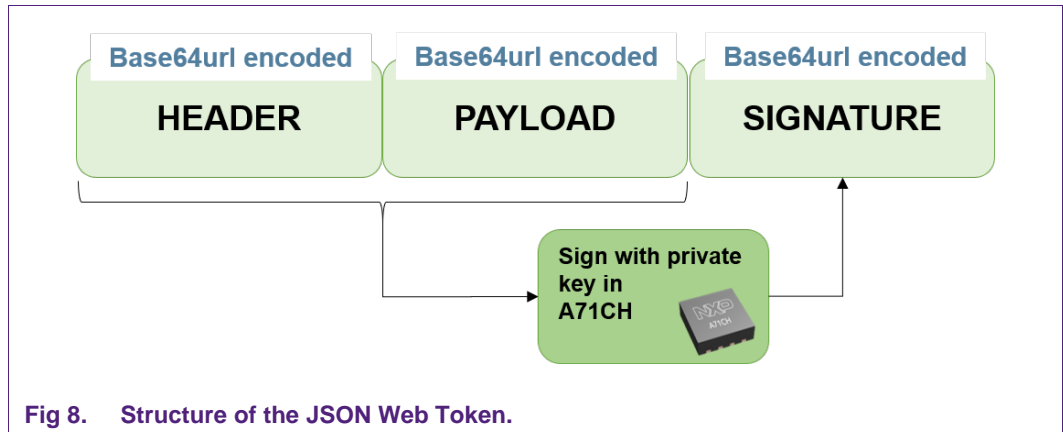


Fig 8. Structure of the JSON Web Token.

Once the signature has been computed and added, the JWT must be set in the password field of the CONNECT message of the MQTT bridge.

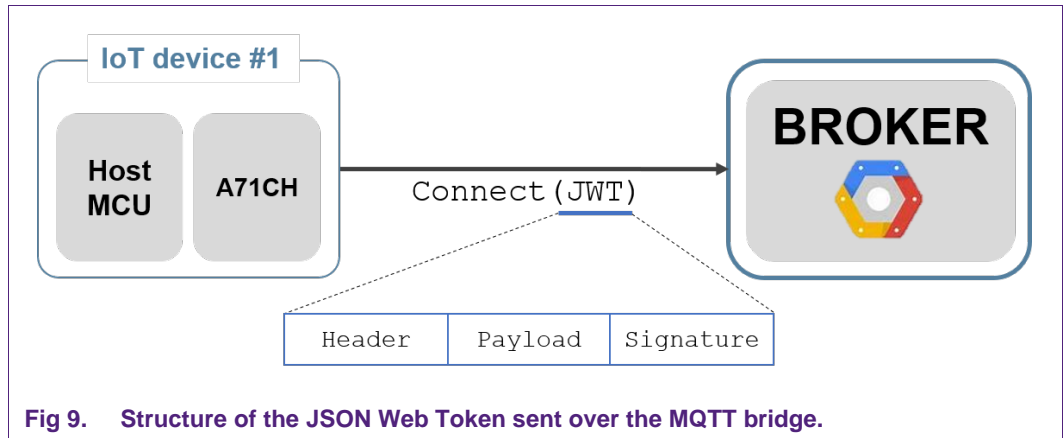


Fig 9. Structure of the JSON Web Token sent over the MQTT bridge.

In [JWT_LIBS], many libraries for Token Signing/Verification can be found.

3.4.2 MQTT Bridge

Message Queue Telemetry Transport (MQTT) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe based communication protocol used in the Internet of Things. It is a low-power and low-bandwidth protocol oriented to embedded devices with small computational resources. More information can be found in [MQTT_LIB].

MQTT protocol follows a star topology in which there is a central node, also called Broker, and a series of clients. The broker device is in charge of managing the network message exchange, while the clients periodically transmit messages and wait for the response.

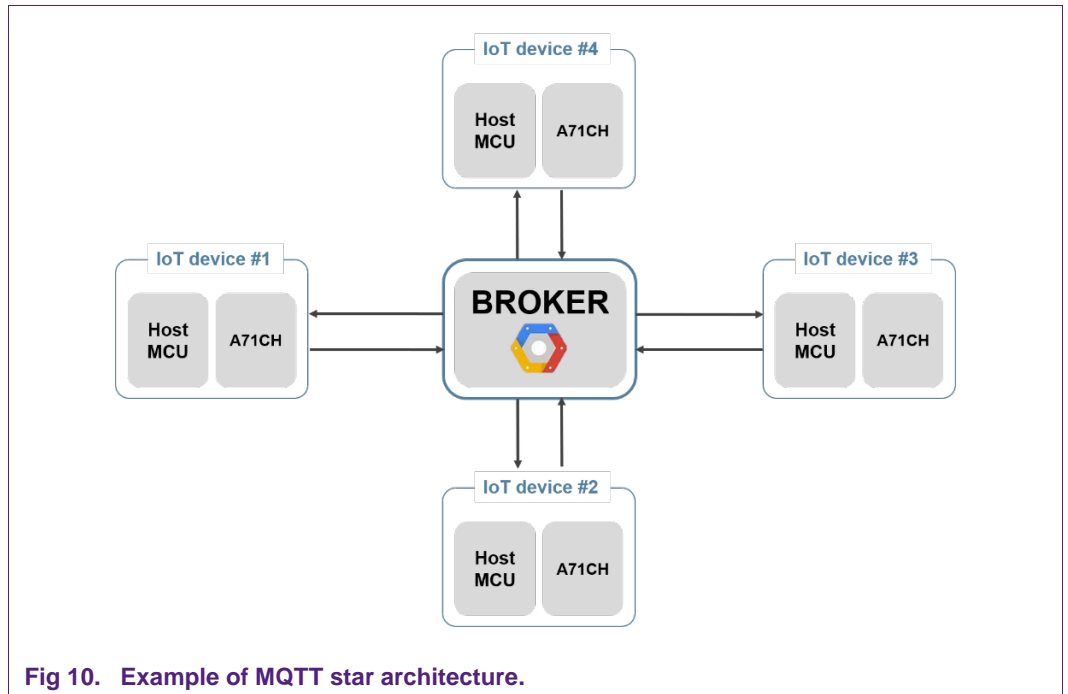


Fig 10. Example of MQTT star architecture.

An MQTT Client will be created to establish a TLS link between the IoT device and Google Cloud IoT Core. When the MQTT client is created, the Client ID must be set to the full path of the device.

```

"""Create our MQTT client. The client_id is a unique string that identifies
this device. For Google Cloud IoT Core, it must be in the format below."""
client = mqtt.Client(
    client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(
        project_id,
        cloud_region,
        registry_id,
        device_id))
    
```

Fig 11. MQTT client creation.

The username field will be set to an *'unused'* string and the password field must contain the JWT previously created to authorize the device. Then, the TLS connection is established, and the device can connect to the MQTT bridge.

```
# With Google Cloud IoT Core, the username field is ignored, and the
# password field is used to transmit a JWT to authorize the device.
client.username_pw_set(
    username='unused',
    password=create_jwt(
        project_id, private_key_file, algorithm))
```

Fig 12. JSON Web Token creation and injection in the password field.

4. Trusted connection to Google Cloud IoT Core using A71CH Customer Programmable type and FRDM-K64F demo application

This section details how to use an A71CH Arduino compatible development kit, a FRDM-K64F board and a demo application to illustrate how to prepare an A71CH Customer Programmable type for connection to Google Cloud IoT Core.

4.1 Hardware setup.

The hardware setup for running this demo consists of:

- FRDM-K64F: Freedom Development Platform for Kinetis K64 with Internet connectivity via Ethernet. This board will act as the IoT device and will connect to the A71CH through the OM3710/A71CHARD Arduino shield.
- OM3710/A71CHARD: Arduino development kit containing a mini PCB board with the A71CH security IC and an Arduino shield compatible with the FRDM-K64F.
- Development PC: A Windows platform will be used to configure and prepare the Google Account and the Google Cloud IoT Core and register a logical device. Additionally, the FRDM-K64F will be programmed with the Development PC using MCUXpresso.

Fig 13 shows the setup that will be used for this demonstration.

Note: The following description will re-use several of the commands already presented in this document; it is provided only for demonstration. Therefore, the subsequent procedure must be adapted and adjusted accordingly for commercial deployment.

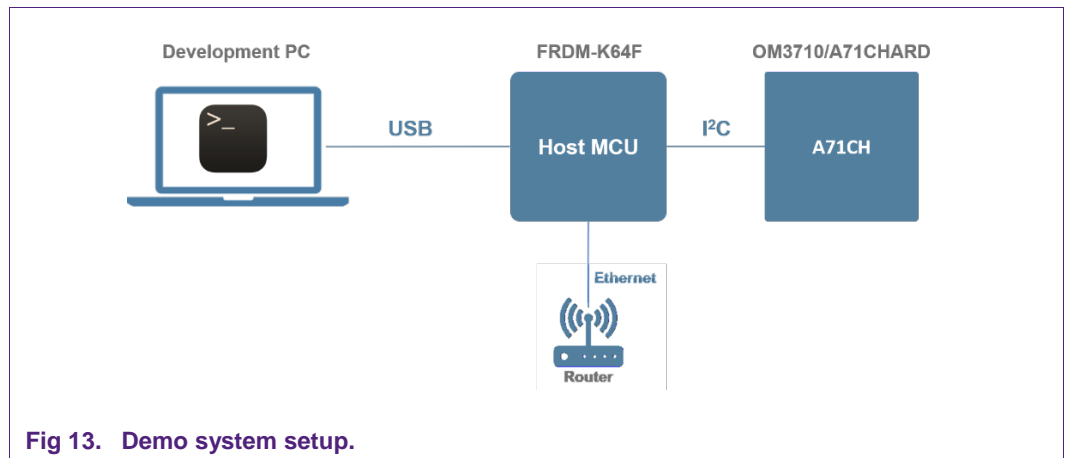


Fig 13. Demo system setup.

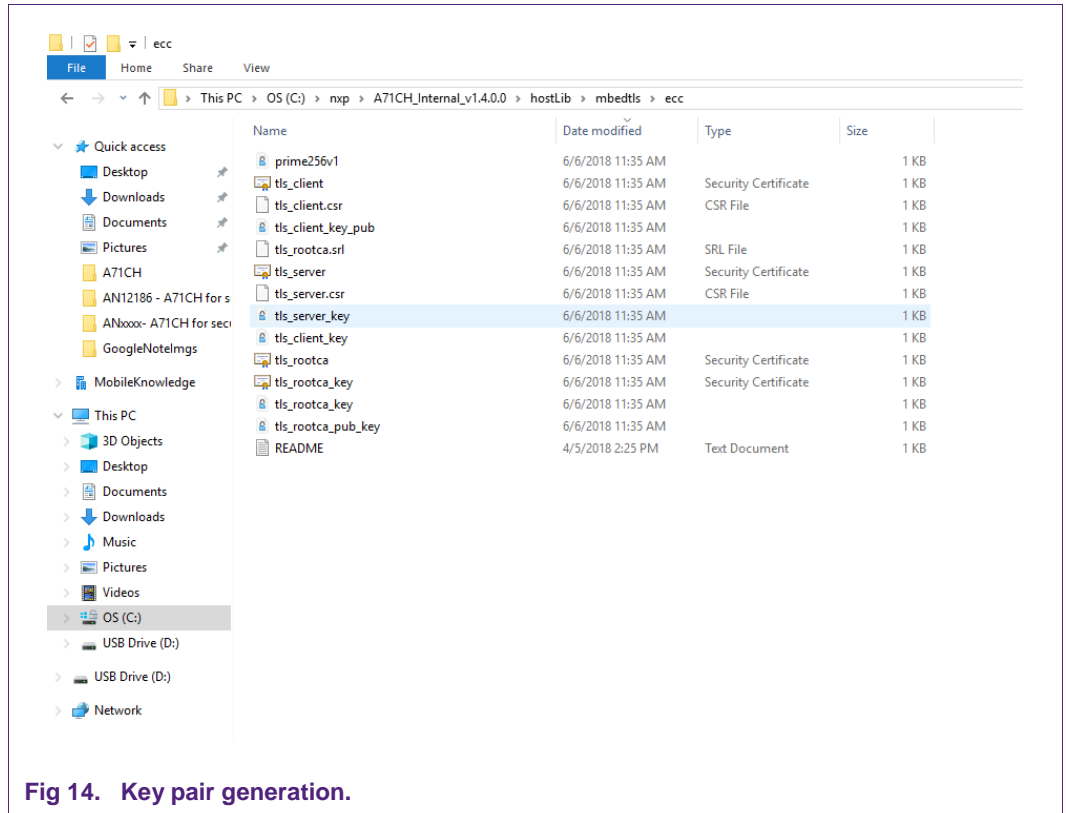
The steps to run this demo are:

1. Create IoT device credentials using OpenSSL
2. Inject keys in A71CH.
3. Set up Google Account and select/create project.
4. Create registries and logical devices.
5. Set up and run the demo.

Note: The software necessary to run this demo is available in the Release 1.4.2 (or later) of the A71CH Host Software Package.

4.2 Create IoT device credentials.

The public/private key pair will be generated using the Development PC. Later they will be securely stored in the A71CH security IC. To create the credentials of the device, you need to run the *RunOnce_CreateCertificate.bat* file (located in the folder `/hostLib/mbedtls/scripts`). Once this file is run, the credentials of the device are created in the `/hostLib/mbedtls/ecc` folder.



4.3 Download keys on the A71CH Security IC

The first step is to connect the FRDM-K64F board to the development PC with a mini USB cable through the OpenSDA port. Using MCUXpresso, you need to import and build the 'vcomA71CH' project (located in `frdmk64f_projects` folder).

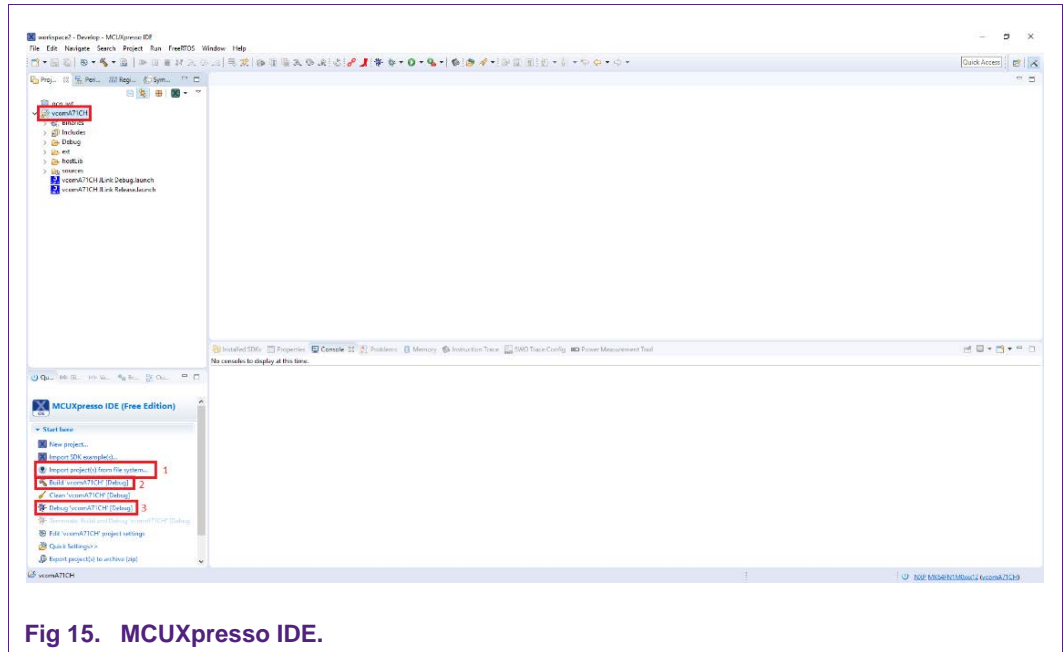


Fig 15. MCUXpresso IDE.

Then, connect the K64F mini USB port to the development PC and check the port name assigned to the board (Virtual Com Port). The setup to download the keys on the A71CH is shown in Fig 16.

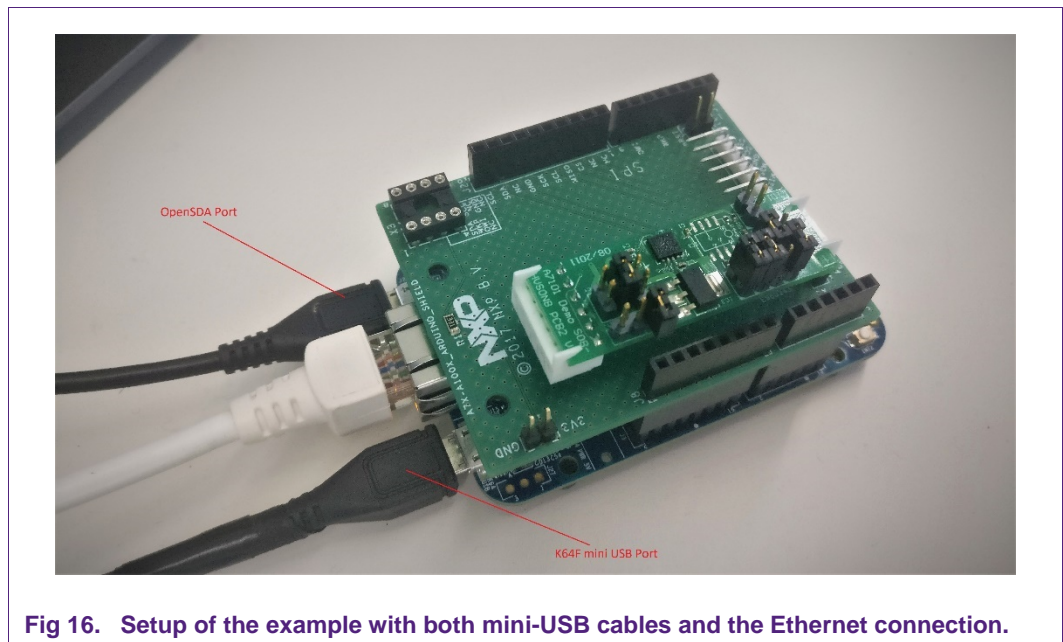


Fig 16. Setup of the example with both mini-USB cables and the Ethernet connection.

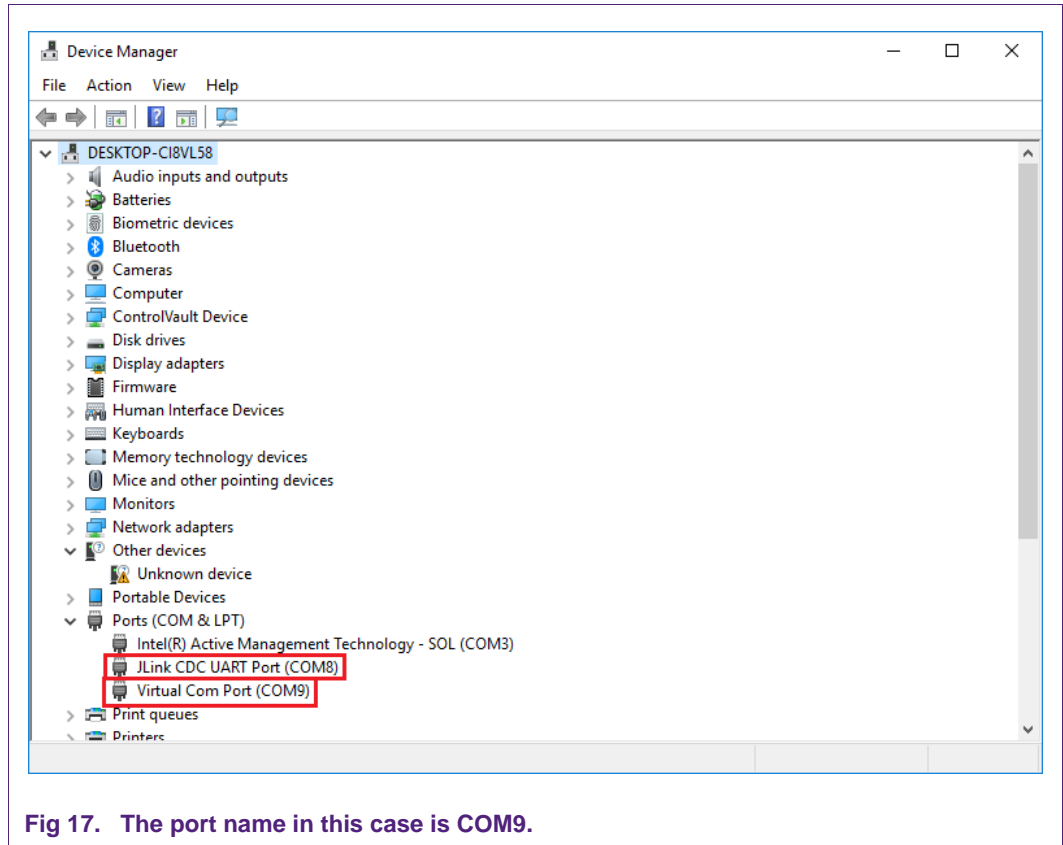


Fig 17. The port name in this case is COM9.

Using Windows Command Prompt, run the batch file `ResetAndUpdateA71CH.bat` with the Windows command line, specifying the port name. This file is in the folder `hostLib/mbedTLS/scripts/`.



Fig 18. Execution of the `ResetAndUpdateA71CH.bat`.

This file will reset and insert the keys and the certificates to the A71CH security IC. The cmd tool will show the different keys being injected.

4.4 Sign in to Google Account and select/create a project.

Sign in to your Google Cloud IoT Core account to create a new project using the following command:

```
$ gcloud auth login
```

Your web browser will open, and you will be able to log in to your Google Account. Once you log in, the active project will be the last one you have worked on; in our case, it is the project with ID `a71ch-test-205010`. As shown in Fig 19, you can choose another project using its `PROJECT_ID`.

```
$ gcloud config set project PROJECT_ID
```

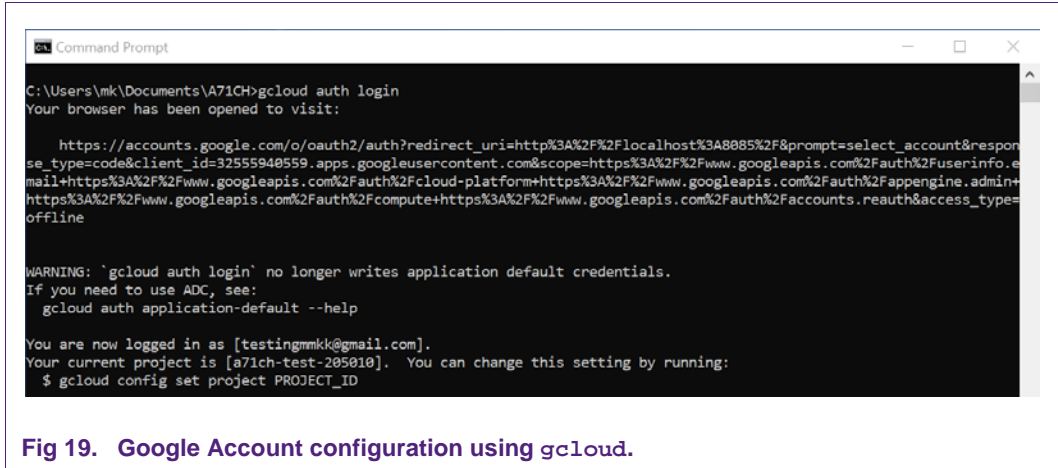


Fig 19. Google Account configuration using gcloud.

In case you want to create a new project, run the `gcloud projects create` command and provide a `PROJECT_ID`, as in Fig 20, where a project named `a71ch-testing-2018` is created.

```
$ gcloud projects create PROJECT_ID
```

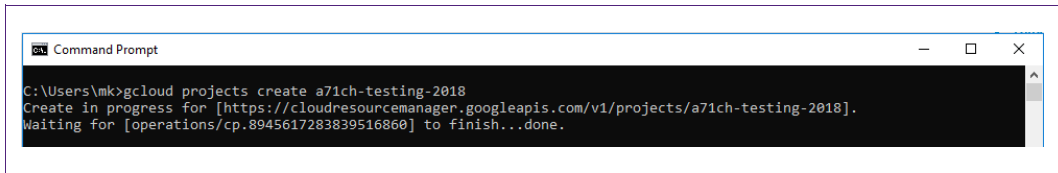


Fig 20. Project creation command.

Once the project has been created, create a Pub/Sub topic with the following command:

```
$ gcloud pubsub topics create myTopic
```

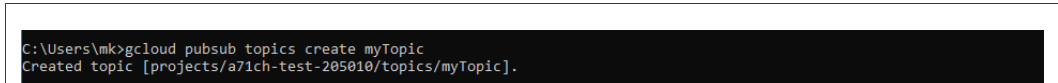


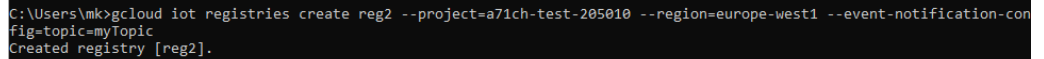
Fig 21. Creation of Pub/Sub topic command.

4.5 Create a device registry and logical device.

The Google Cloud IoT Core Device Manager lets you create and configure device registries and the devices within them. In Fig 22, the creation of a registry named `reg2`

```
$ gcloud iot registries create reg2 --project=PROJECT_ID --  
region=europe-west1 --event-notification-config=topic=myTopic
```

can be observed.

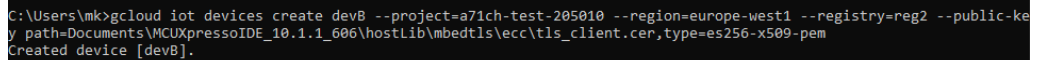


```
C:\Users\mk>gcloud iot registries create reg2 --project=a71ch-test-205010 --region=europe-west1 --event-notification-con  
fig=topic=myTopic  
Created registry [reg2].
```

Fig 22. Creation of a device registry using gcloud.

When a logical device is created inside a registry, the device is defined as a Google Cloud IoT Core resource. A device is created using the public key or the certificate that has been previously generated. In Fig 23, the registration of a device identified as `devB` is shown. As can be observed, a certificate is uploaded (`tls_client.cer`) and the type flag is defined as `es256-x509-pem`.

```
$ gcloud iot devices create devB --project=PROJECT_ID --  
region=europe-west1 --registry=reg2 --public-key  
path=PATH_TO_YOUR_KEY,type=es256-x509-pem
```



```
C:\Users\mk>gcloud iot devices create devB --project=a71ch-test-205010 --region=europe-west1 --registry=reg2 --public-ke  
y path=Documents\MCUxpressoIDE_10.1.1_606\hostLib\mbedtls\ecc\tls_client.cer,type=es256-x509-pem  
Created device [devB].
```

Fig 23. Device registration using gcloud.

Once the device is created, you can check that it has been correctly registered in the GCP Console, as in Fig 24, through the 'Device details' page.

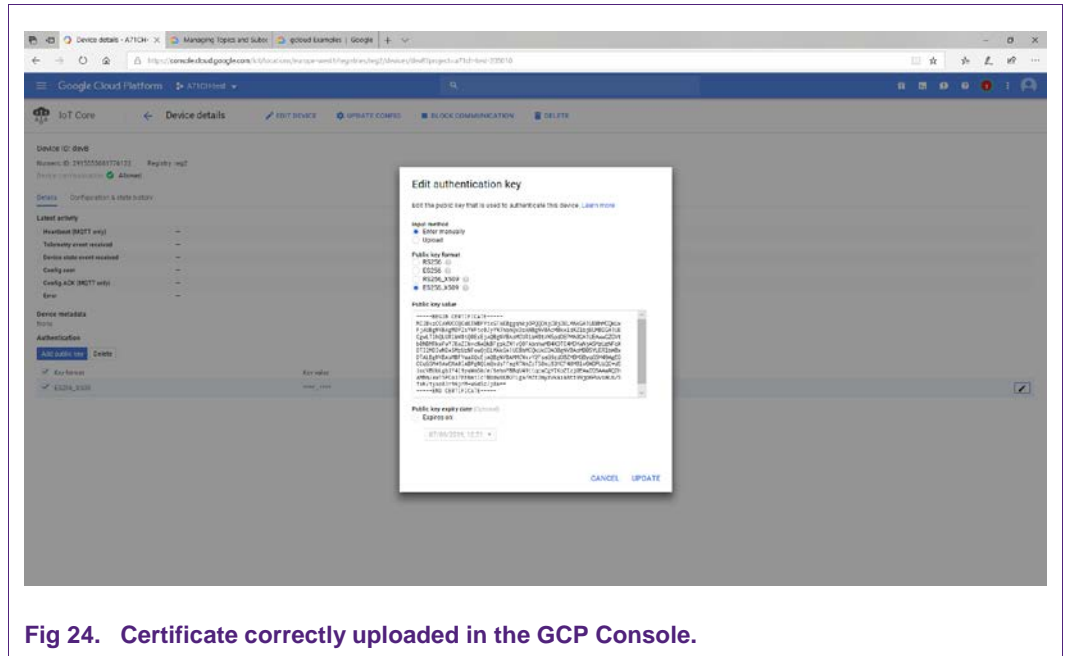


Fig 24. Certificate correctly uploaded in the GCP Console.

4.6 Set up and run the demo.

Before running the demo that connects the IoT device to Google Cloud IoT Core, define the names of the project, the registry, the device and the location. Update the 'gcp_iot_config.h' file and change the definitions shown below.

```

28 #define GCP_PROJECT_NAME "a71ch-test-205010"
29 #define GCP_LOCATION_NAME "europe-west1"
30 #define GCP_REGISTRY_NAME "reg2"
31 #define GCP_DEVICE_NAME "devB"
    
```

Fig 25. Definitions that require being updated.

Once the file has been updated, build and run the 'gcp_jwt' project using MCUXpresso (also located in frdmk64f_projects folder). If all the previous steps have been correctly performed, the FRDM-K64F board will be able to correctly connect to Google Cloud IoT Core. If you check the 'Device details' page of the GCP Console, you can observe the details of the connection of the IoT Device. In Fig 26, the activity of the IoT Device can be observed.

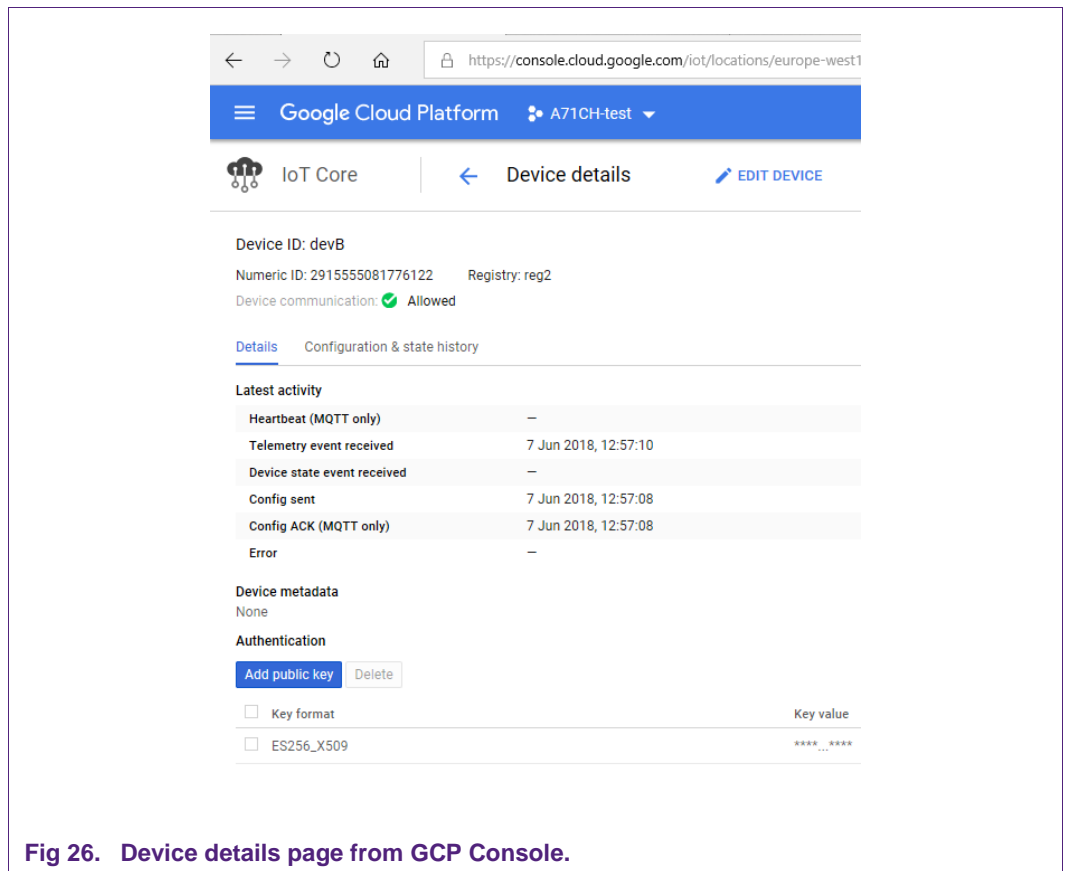


Fig 26. Device details page from GCP Console.

From the Dashboard of the project, you can access the statistics of request that the API has encountered. In Fig 27, the APIs & Services page can be observed. It provides information concerning the requests to the API, the errors encountered and the latency of the requests.

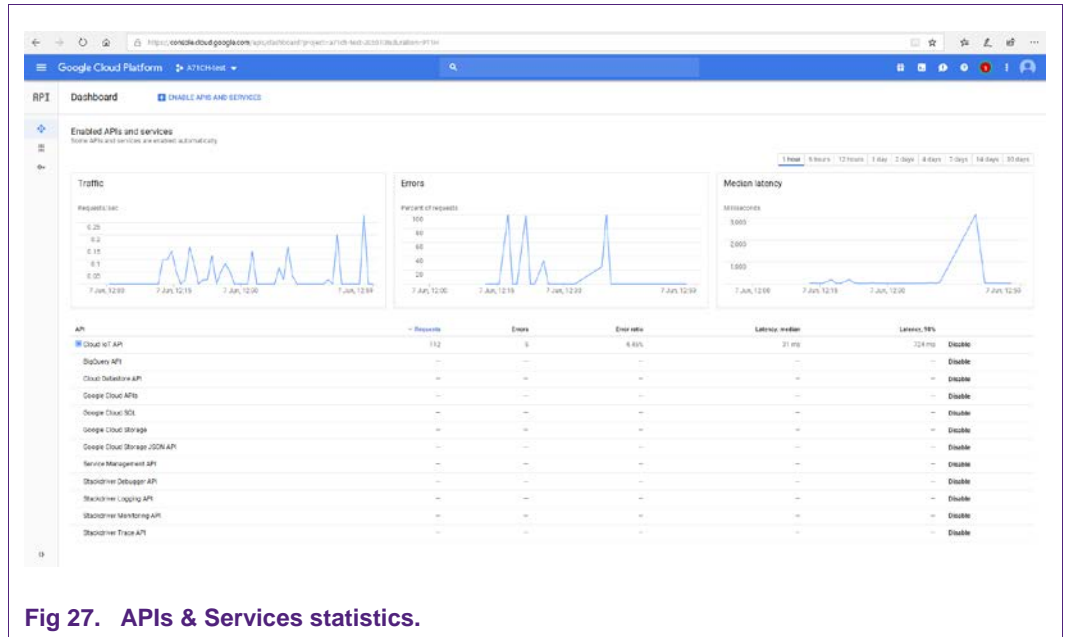


Fig 27. APIs & Services statistics.

5. Trusted connection to Google Cloud IoT Core using A71CH Customer Programmable type and i.MX6UltraLite demo application

This section details how to use an A71CH Arduino compatible development kit, an i.MX6UltraLite board and a demo application to illustrate how to prepare an A71CH Customer Programmable type for connection to Google Cloud IoT Core.

5.1 Hardware setup

The hardware setup for running this demo consists of:

- MCIMX6UL-EVK: i.MX6UltraLite MCU evaluation board with Internet connectivity via Ethernet. This board will act as the IoT device and will connect to the A71CH through OM3710/A71CHARD Arduino shield.
- OM3710/A71CHARD: Arduino development kit containing a mini PCB board with the A71CH security IC and an Arduino shield compatible with the FRDM-K64F.
- Development PC: A Windows platform will be used to configure and prepare the Google Account and the Google Cloud IoT Core and register a logical device. Additionally, the FRDM-K64F will be programmed with the Development PC using MCUXpresso.

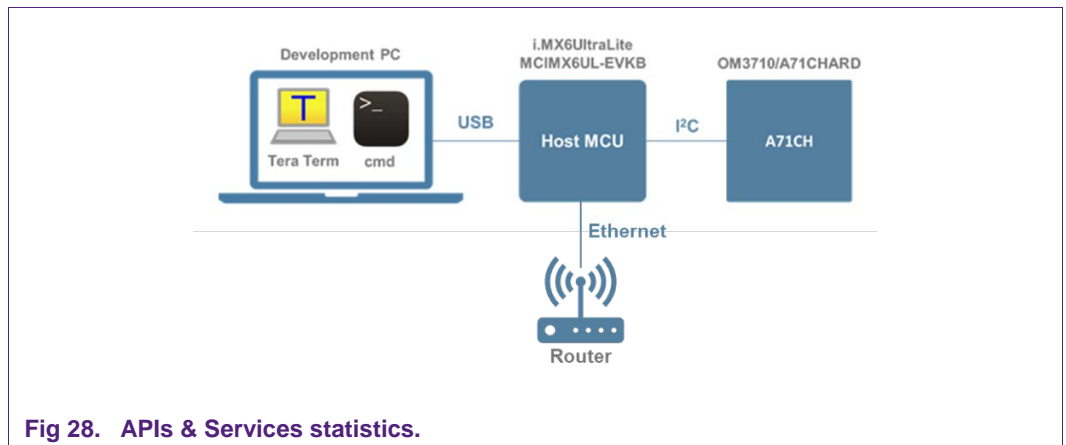


Fig 28. APIs & Services statistics.

It is assumed that the A71CH is already provisioned. The corresponding certificate or public key must be uploaded to Google Cloud, as shown in Section 3.3.2. The steps to run this demo are:

1. Retrieve reference keys from A71CH.
2. Retrieve root certificate of Google Cloud Platform.
3. Download source code.
4. Update source code.
5. Compile and run demonstration.

5.2 Retrieve reference keys from A71CH

The first step is to create a directory where the files and keys necessary for the Google Cloud IoT Core demonstration will be stored. Run the following commands:

```
$ mkdir -p ~/gcp
$ mkdir -p ~/gcp/keys
```

Once the directories have been created, the private key must be obtained from the A71CH security IC and stored in the `gcp/keys` directory. This command will create a reference to the key pair (`-c 10`) in slot 0 (`-x 0`) and store it in the desired path (`-r ~/gcp/keys/tls_client_refkey.pem`).

```
$ ~/axHostSw/linux/a71chConfig_i2c_imx refpem -c 10 -x 0 -r
~/gcp/keys/tls_client_refkey.pem
```

5.3 Retrieve root certificate of Google Cloud Platform

The root certificate of Google Cloud IoT Core can be downloaded from [GCP_ROOT]. To do so from i.MX6UL board, run the following command. It will be stored in the `gcp/keys` directory.

```
$ curl -o ~/gcp/keys/googleroots.pem
https://pki.google.com/roots.pem
```

Once the certificate has been downloaded, the contents of the keys folder should look as shown in Fig 29.

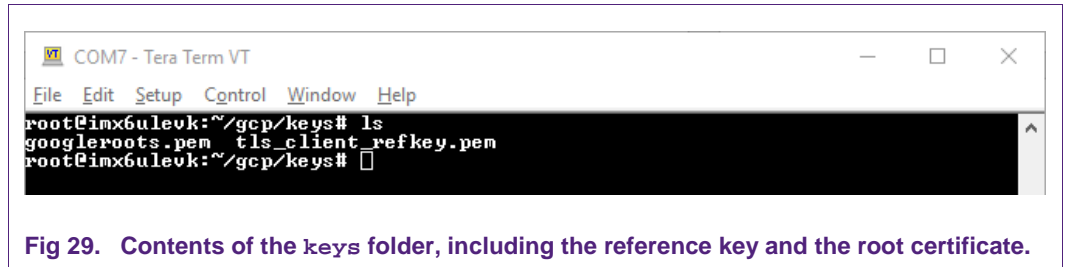


Fig 29. Contents of the keys folder, including the reference key and the root certificate.

5.4 Download source code

The source code needed for the demo can be downloaded from four different GitHub repositories. The commands below will download the code to the gcp folder.

```
$ cd ~/gcp
$ git clone https://github.com/akheron/jansson.git
$ git clone https://github.com/benmcollins/libjwt.git
$ git clone https://github.com/eclipse/paho.mqtt.c.git
$ git clone https://github.com/GoogleCloudPlatform/cpp-docs-samples.git
```

If all the repositories have been correctly downloaded, the contents of the gcp directory should appear as in Fig 30.

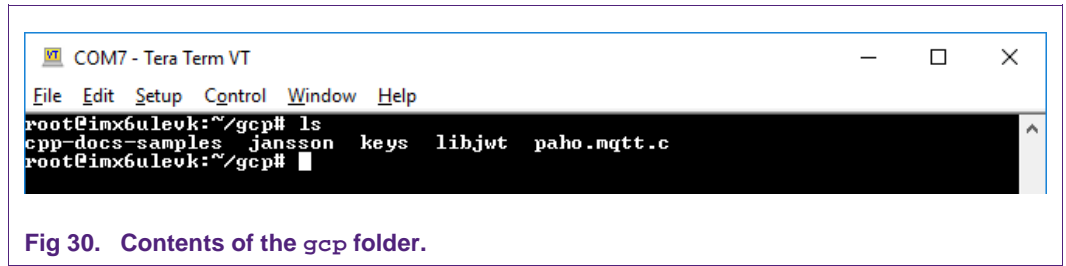


Fig 30. Contents of the gcp folder.

5.5 Update source code.

The code downloaded from the cpp-docs-samples repository cannot use OpenSSL Engine directly. Thus, a few modifications are necessary in the code.

Note: The simple text editor *vi* will be used to edit the source code. Three commands are necessary:

- 'vi <filename>': Open the file with the text editor.
- 'i': Start editing the file.
- Esc + ':wq': Save the file and exit the text editor.

5.5.1 Update Makefile

Path	~/gcp/cpp-docs-samples/iot/mqtt-ciots/Makefile
------	--

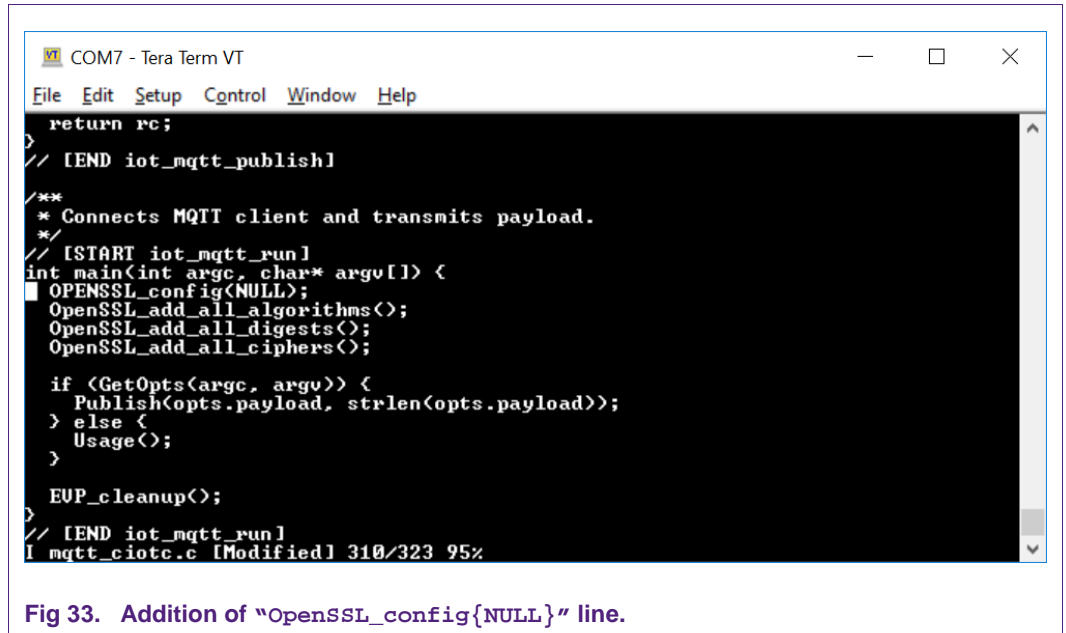


Fig 33. Addition of "OPENSSL_config{NULL}" line.

Next, define the topic where the messages will be published. Type the full address of the topic created in Section 4.4 in the ".topic" option, as shown in Fig 34.

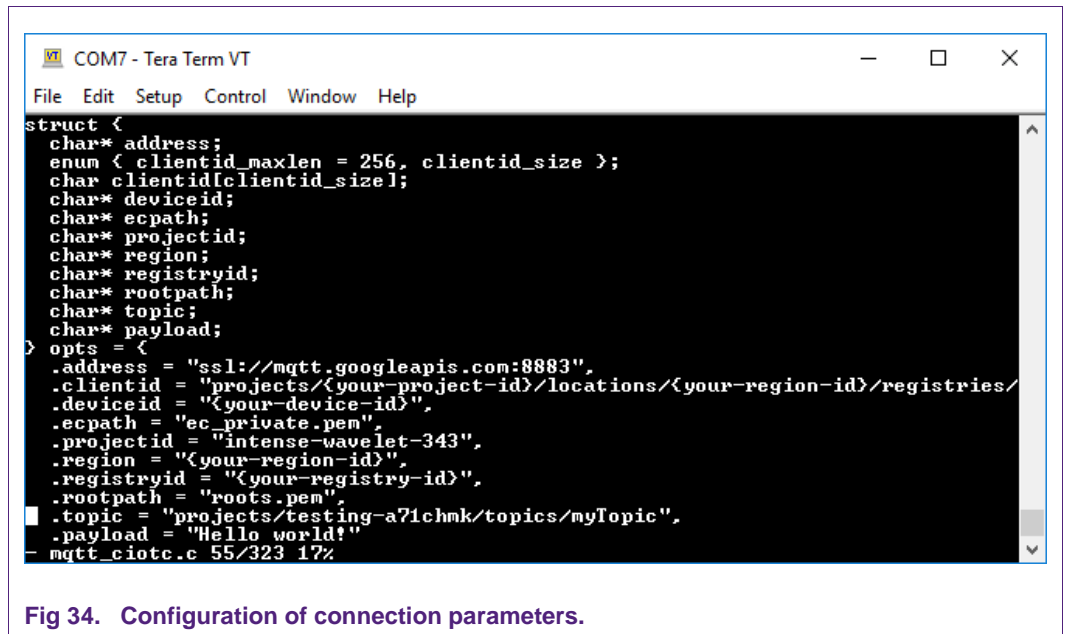


Fig 34. Configuration of connection parameters.

5.6 Compile and run demonstration

Once the changes have been applied, compile the different projects that have been downloaded. Run the following commands:

```
$ cd ~/gcp/jansson
$ autoreconf -i
$ ./configure
$ make all
$ sudo make install

$ cd ~/gcp/libjwt
$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
$ autoreconf -i
$ ./configure
$ make all
$ sudo make install

$ cd ~/gcp/paho.mqtt.c
$ make all
$ sudo make install
$ ldconfig /usr/local/lib

$ cd ~/gcp/cpp-docs-samples/iot/mqtt-ciots
$ make all
```

To run the demo, create a script like the one shown below. The script can be created using *vi*.

```
$ vi sh_run.sh

#!/bin/sh
OPENSSL_CONF=/etc/ssl/opensslA71CH_i2c.cnf

export OPENSSL_CONF

./mqtt-ciots "{ \"msg\" : \"$1\" }" \
--deviceid Dev2 \
--region europe-west1 \
--registryid registry2 \
--projectid testing-a71chmk \
--ecpath ~/gcp/keys/tls_client_refkey.pem \
--rootpath ~/gcp/keys/googleroots.pem
```

Once the script is created with your connection parameters, run the code with the next command. To be able to run the script, add the execution permission:

```
$ chmod +x sh_script.sh
$ ./sh_run.sh hello
```

A success message of the publishing event will be shown in the console:

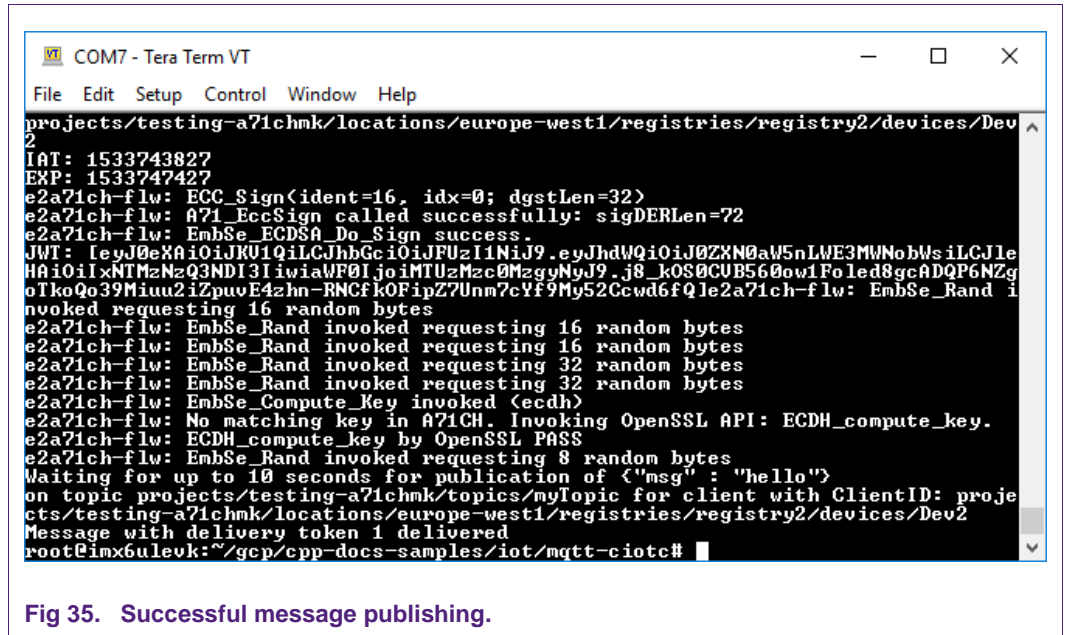


Fig 35. Successful message publishing.

The activity of the IoT device can be observed through the Cloud IoT API traffic, as shown in Fig 36.

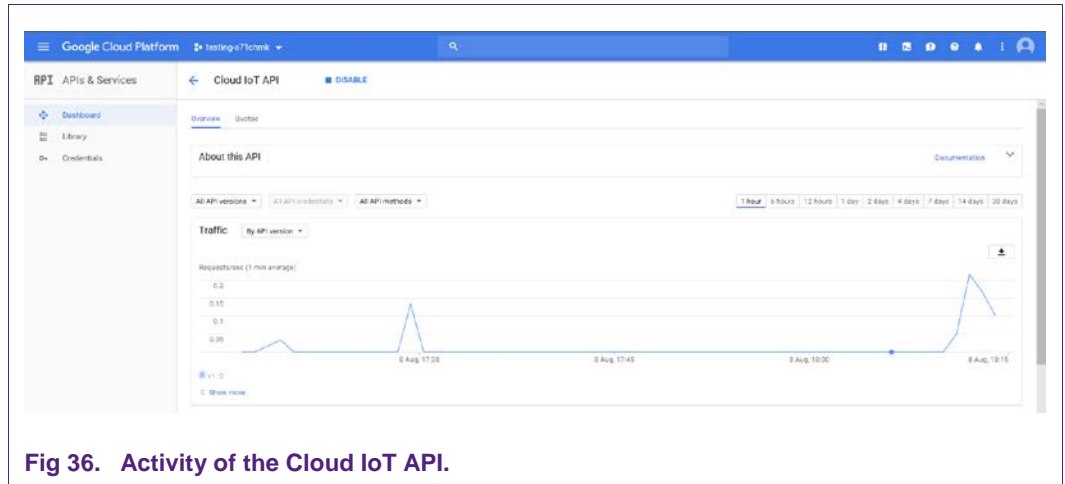


Fig 36. Activity of the Cloud IoT API.

6. A71CH Trust Provisioning options

Key and credentials generation is a relatively complex process and can introduce vulnerabilities if not done properly. Manual provisioning can lead to errors and is difficult to scale when more devices are needed. Also, to ensure that keys are kept safe, injection should take place in a trusted environment; in a facility with security features such as tightly controlled access, careful personnel screening, and secure IT systems that protect against cyber-attacks and theft of credentials. NXP offers three options for the A71CH Trust Provisioning:

- NXP's Trust Provisioning, available for high volume orders of more than 150K units.

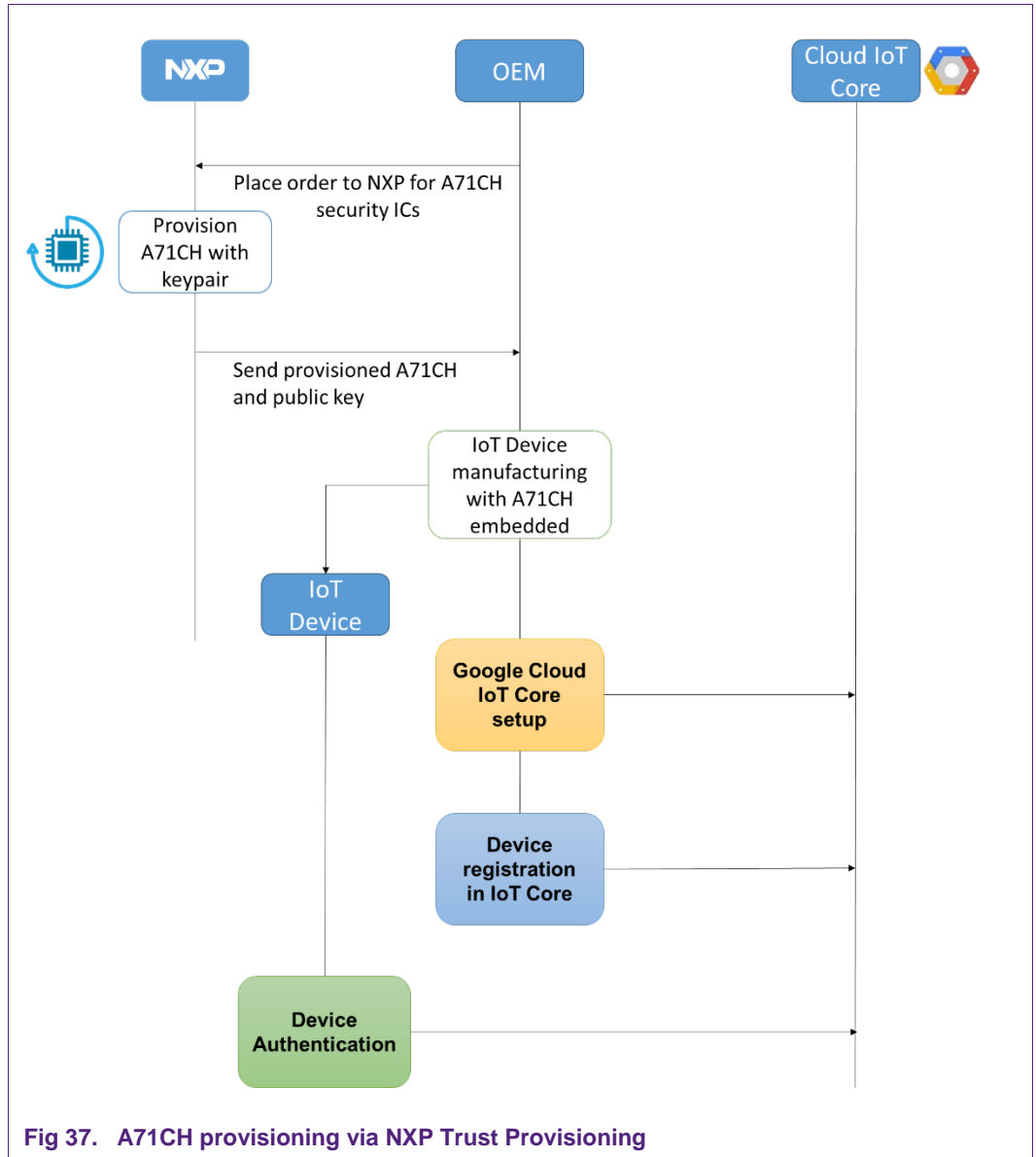
- Trust Provisioning via distributors and third-party partners in programming centers, available for orders of any size.
- The A71CH Configure tool, allowing the injection of credentials into the A71CH for evaluation, development and testing purposes.

In this section, the NXP Trust Provisioning and the Trust Provisioning via distributors and third-party partner flows are explained.

6.1 NXP Trust Provisioning

The NXP Trust Provisioning provisions die-individual credentials on behalf of your organization. As such, your organization does not need to take care of the key injection, owning PKI infrastructure or subcontracting a third-party PKI infrastructure for the IC trust provisioning. The flow using the NXP Trust Provisioning can be summarized in the following steps:

1. The OEM places an order to NXP for a certain quantity of A71CH security ICs.
2. NXP provisions the A71CH ICs with individual key pairs.
3. NXP sends the provisioned A71CH and public keys to the OEM.
4. The OEM embeds the A71CH into their IoT devices in the manufacturing process
5. The OEM registers the IoT devices in their Google Cloud IoT Core account (Section 3.3).
6. IoT device can authenticate with Google Cloud IoT Core (Section 3.4).



6.2 Third-party programming facility

In this option, the distributor or third-party programming facility provisions die-individual credentials on behalf of your organization. Fig 38 shows the flow diagram of this process, which can be summarized in the following steps:

1. NXP ships A71CH ICs to the distributor / programming facility
2. The OEM places an order with the distributor / programming facility for the provisioning of A71CH ICs.
3. The distributor / programming facility provisions A71CH ICs with die-individual credentials.

4. The programming facility sends provisioned A71CH and corresponding public keys to OEM.
7. The OEM embeds the A71CH into their IoT devices in the manufacturing process
8. The OEM registers the IoT devices in their Google Cloud IoT Core account (Section 3.3).
5. IoT device can authenticate with Google Cloud IoT Core (Section 3.4).

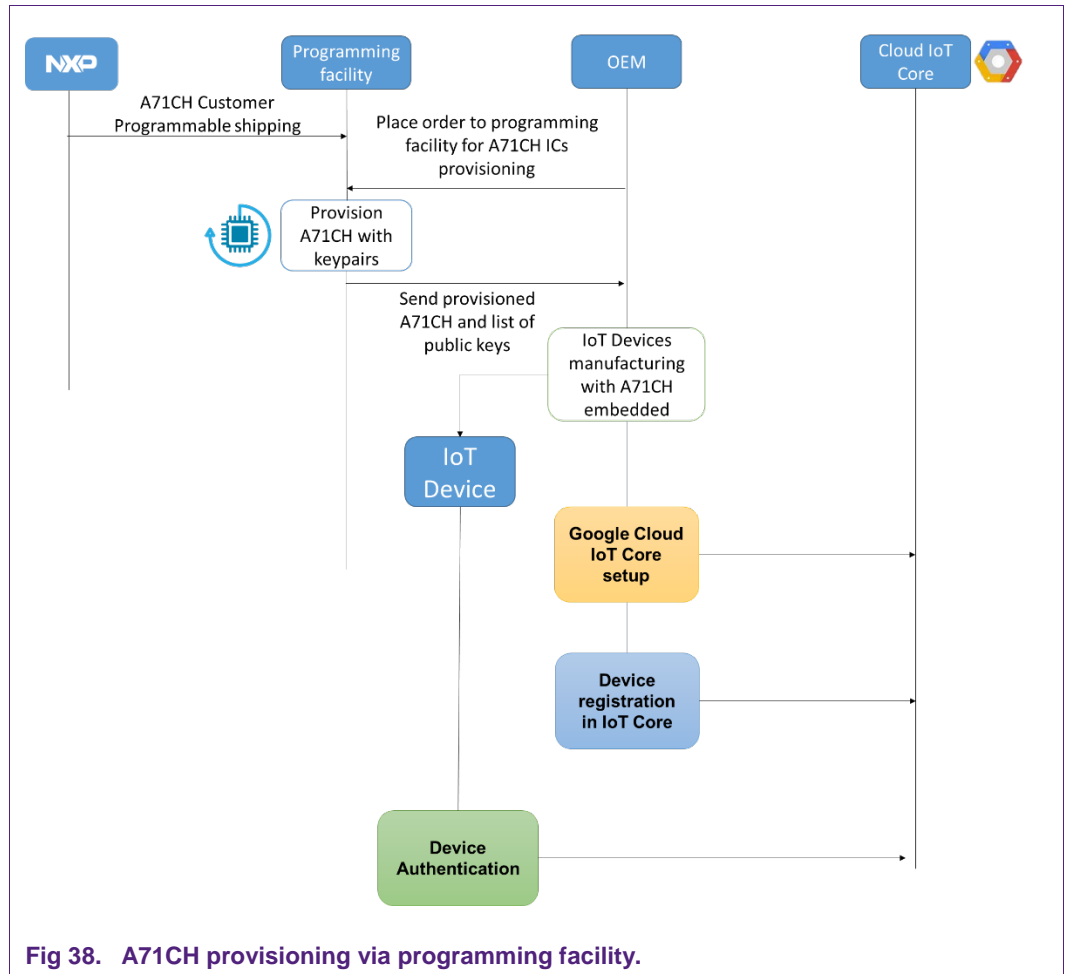


Fig 38. A71CH provisioning via programming facility.

7. Reference documentation

Table 1. Reference documents

[JWT_LIBS]	JSON Web Token Libraries – http://jwt.io
[GCLOUD_API]	Google Cloud API Overview – https://cloud.google.com/sdk/gcloud/reference
[CLOUD_SDK]	Cloud SDK download page – https://cloud.google.com/sdk/
[A71CH_HOST_SW]	A71CH Host Software Package (Windows Installer) – DocStore, document number sw4673** ¹ , Version 01.04.00 (or later), available on www.nxp.com/A71CH A71CH Host Software Package (Bash Installer) – DocStore, document number sw4672**, Version 01.04.00 (or later), available on www.nxp.com/A71CH
[MQTT_LIB]	MQTT Reference – http://mqtt.org
[QUICK_START_IMX6]	AN12119 Quick start guide for OM3710/A71CHARD i.MX6 – Application note, document number 4582**
[GCP_ROOT]	Root certificate of Google Cloud Platform - https://pki.google.com/roots.pem

¹ **...document version number

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the

customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out of the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US\$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.1 Licenses

ICs with DPA Countermeasures functionality



NXP ICs containing functionality implementing countermeasures to Differential Power Analysis and Simple Power Analysis are produced and sold under applicable license from Cryptography Research, Inc.

8.2 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

FabKey — is a trademark of NXP B.V.

IC-bus — logo is a trademark of NXP B.V.

9. List of figures

Fig 1.	Flow diagram of the connection of the A71CH Security IC to Google Cloud.....	4	Fig 36.	Activity of the Cloud IoT API.....	29
Fig 2.	IoT device authentication flow to Google Cloud IoT Core.	5	Fig 37.	A71CH provisioning via NXP Trust Provisioning	31
Fig 3.	IoT device authentication flow to Google Cloud IoT Core using IoT device cert.	6	Fig 38.	A71CH provisioning via programming facility. .	32
Fig 4.	Google Cloud Platform Home Page.....	8			
Fig 5.	'Create device registry' page in GCP.	9			
Fig 6.	'Add device' page.....	10			
Fig 7.	Device authentication with device credentials stored in A71CH.....	11			
Fig 8.	Structure of the JSON Web Token.....	12			
Fig 9.	Structure of the JSON Web Token sent over the MQTT bridge.....	12			
Fig 10.	Example of MQTT star architecture.	13			
Fig 11.	MQTT client creation.....	13			
Fig 12.	JSON Web Token creation and injection in the password field.	14			
Fig 13.	Demo system setup.	15			
Fig 14.	Key pair generation.	16			
Fig 15.	MCUXpresso IDE.....	17			
Fig 16.	Setup of the example with both mini-USB cables and the Ethernet connection.	17			
Fig 17.	The port name in this case is COM9.....	18			
Fig 18.	Execution of the ResetAndUpdateA71CH.bat.	18			
Fig 19.	Google Account configuration using <code>gcloud</code> ..	19			
Fig 20.	Project creation command.	19			
Fig 21.	Creation of Pub/Sub topic command.	19			
Fig 22.	Creation of a device registry using <code>gcloud</code>	20			
Fig 23.	Device registration using <code>gcloud</code>	20			
Fig 24.	Certificate correctly uploaded in the GCP Console.....	21			
Fig 25.	Definitions that require being updated.	21			
Fig 26.	Device details page from GCP Console.....	22			
Fig 27.	APIs & Services statistics.....	23			
Fig 28.	APIs & Services statistics.....	24			
Fig 29.	Contents of the <code>keys</code> folder, including the reference key and the root certificate.....	25			
Fig 30.	Contents of the <code>gcp</code> folder.	25			
Fig 31.	Updated Makefile.	26			
Fig 32.	Addition of "#include <code>openssl/conf.h</code> " line.	26			
Fig 33.	Addition of " <code>OpenSSL_config{NULL}</code> " line.....	27			
Fig 34.	Configuration of connection parameters.	27			
Fig 35.	Successful message publishing.	29			

10. Contents

1.	Introduction	3	5.	Trusted connection to Google Cloud IoT Core using A71CH Customer Programmable type and i.MX6UltraLite demo application	23
2.	Google Cloud IoT Core setup.....	3	5.1	Hardware setup	23
2.1	Google Cloud IoT Core setup.....	4	5.2	Retrieve reference keys from A71CH.....	24
2.2	IoT device registration in IoT Core	4	5.3	Retrieve root certificate of Google Cloud Platform	24
2.3	IoT device authentication	4	5.4	Download source code.....	25
3.	Trusted connection to Google Cloud IoT Core process	7	5.5	Update source code.....	25
3.1	Creation of credentials	7	5.5.1	Update Makefile	25
3.2	Google Cloud IoT Core setup.....	7	5.5.2	Update mqtt_ciotc.c.....	26
3.2.1	Google Account login and project selection/creation.....	7	5.6	Compile and run demonstration	28
3.2.1.1	Using GCP Console	7	6.	A71CH Trust Provisioning options	29
3.2.1.2	Using gcloud.....	8	6.1	NXP Trust Provisioning	30
3.3	IoT device registration in Google Cloud IoT Core	8	6.2	Third-party programming facility	31
3.3.1	Create a device registry	8	7.	Reference documentation.....	33
3.3.1.1	Device registry creation using GCP Console	9	8.	Legal information	34
3.3.1.2	Device registry creation using gcloud	9	8.1	Definitions.....	34
3.3.2	Create a logical device.....	10	8.2	Disclaimers.....	34
3.3.2.1	Logical device creation using GCP Console	10	8.1	Licenses	34
3.3.2.2	Logical device creation using gcloud	10	8.2	Trademarks	34
3.4	Device authentication.....	11	9.	List of figures.....	35
3.4.1	JSON Web Tokens	11	10.	Contents	36
3.4.2	MQTT Bridge.....	13			
4.	Trusted connection to Google Cloud IoT Core using A71CH Customer Programmable type and FRDM-K64F demo application	14			
4.1	Hardware setup.....	14			
4.2	Create IoT device credentials.....	15			
4.3	Download keys on the A71CH Security IC	16			
4.4	Sign in to Google Account and select/create a project.....	18			
4.5	Create a device registry and logical device.....	20			
4.6	Set up and run the demo.....	21			

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.