This document describes how the A71CH security IC can be used to establish a secure connection with an AWS.
Contact information

For more information, please visit: http://www.nxp.com
1. Introduction

This document describes how the A71CH Security IC can be used to establish a secure connection between an IoT device and Amazon Web Services (AWS) IoT cloud. The connection between the IoT device and AWS IoT cloud will be established with the MQTT protocol [MQTT], running across Transport Layer Security (TLS) protocol.

To establish a Transport Layer Security (TLS) connection, the IoT device certificate has to be registered in AWS IoT. The AWS Just-in-time registration (JITR) mechanism allows the IoT device to automatically register its digital certificate, thus establishing a connection with AWS IoT. This document contains a brief introduction to TLS and key security concepts and then presents the workflow to prepare the JITR of an IoT device certificate in AWS IoT.

2. A71CH overview

The A71CH is a ready-to-use solution enabling ease-of-use security services for the IoT device makers. It is 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 provides an outstanding level of security measures protecting the IC against physical and logical attacks. It can be used with various host platforms and host operating systems to secure a broad range of applications. In addition, it is complemented by a comprehensive product support package, offering easy design-in with plug & play host application code, easy-to-use development kits, reference designs, documentation and IC samples for product evaluation.

3. Public key infrastructure and ECC fundamentals

Security is an essential requirement for any IoT design. Thus, security should not be considered as differentiator option but rather a standard feature for the IoT designers. IoT devices must follow a secure-by-design approach, ensuring secure storage of credentials, device authentication, secure code execution and safe connections to remote servers among others. In this security context, the A71CH security IC is designed specifically to offer protected access to credentials, secure connection to private or public clouds and cryptographic device proof-of-origin verification.

Asymmetric cryptography, also known as public key cryptography, is any cryptographic algorithms based on a pair of keys: a public key and a private key. The private key must be kept secret, while the public key can be shared.

RSA (Rivest, Shamir and Adleman) and Elliptic-Curve Cryptography (ECC) are two of the most widely used asymmetric cryptography algorithms. In the case of ECC cryptography, it is based on the algebraic structure of elliptic curves over finite fields. Therefore, each key pair (public and private key) is generated from a certain elliptical curve.

The digital signature, digital certificates, Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm are briefly explained in the next sections.
3.1 Digital signature

A digital signature is used to guarantee the authenticity, the integrity and non-repudiation of a message. A signing algorithm generates a signature given a message and a private key. A signature verifying algorithm accepts or rejects a message given the public key and the signature.

Fig 1 illustrates an example of digital signature. In this case, the message is signed with the sender private key. The receiver will validate the signature using both the message and the sender public.

![Digital signature diagram](image)

3.2 Digital certificate, Certification Authority (CA) and Certificate Signing Request (CSR)

Digital certificates are used to prove the authenticity of shared public keys. Digital certificates are electronic documents that include information about the sender public key, identity of its owner and the signature of a trusted entity that has verified the contents of the certificate, normally called Certificate Authority (CA).

A Certificate Authority (CA) is an entity that issues digital certificates. The CA is trusted by both the certificate sender and the certificate receiver, and it is typically in charge of receiving a Certificate Signing Request (CSR) and generating a new certificate based upon information contained in the CSR and signed with the CA private key.

Therefore, a CSR is a request that contains all the necessary information, e.g., sender public key and relevant information to generate a new digital certificate.

Fig 2 shows digital certificates generation steps. First, the interested device (sender) creates a Certificate Signing Request. The CSR is then sent to the CA and a new digital certificate is created and signed with the CA private key. Also, the basic contents of this new digital certificate are illustrated in the figure.
3.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm uses ECC to provide a variant of the Digital Signature Algorithm (DSA). A pair of keys (public and private) are generated from an elliptic curve, and these can be used both for signing or verifying a message’s signature. Fig 3 illustrates an example of ECDSA application. In this example, the sender device generates a signature with its private key. The signed message is sent together with the sender digital certificate to the receiver. Finally, the receiver retrieves the sender public key from the digital certificate and uses it to validate the signature of the received message.

3.4 Elliptic Curve Diffie-Hellman (ECDH)

Elliptic Curve Diffie-Hellman algorithm (ECDH) is a key-agreement protocol. The goal of ECDH is to reach a key agreement between two parties, each having an elliptic-curve key pair generated from the same domain parameters. When the agreement has been reached, a shared secret key, usually referred to as the ‘master key’, is derived to obtain
session keys. These session keys will be employed to establish a communication using symmetric-key encryption algorithms.

The sender and the receiver have its own elliptical key pair. Both the sender and receiver public keys are shared with each other. In this case, the exchange has been represented with digital certificates. Each party can compute the secret key using their own private key and the public key obtained from the received certificate. Due to the elliptical curve properties and the fact that both key pairs have been generated from the same domain parameters, the computed secret key is the same for both parties. This common secret key will be further used for establishing a communication and encrypt messages based on symmetrical cryptography. Fig 4 illustrates the usage of ECDH for a shared secret key agreement.

In the Elliptic-curve Diffie-Hellman Ephemeral (ECDHE) algorithm case, a new elliptical key pair is generated for each key agreement instead of sharing the already existing public keys.

3.5 A71CH ECC supported functionality

The A71CH security IC supports the following ECC functionality:

- Signature generation and verification (ECDSA).
- Shared secret calculation using Key Agreement (ECDH or ECDH-E).
- Secure storage, generation, insertion or deletion of key pairs (NIST-P256 elliptical curve).
4. A71CH for secure connection to AWS IoT cloud

The scope of this document is to explain step by step how to connect an IoT device to an AWS IoT cloud by using the available AWS IoT services.

The IoT device will feature an A71CH Security IC to securely store credentials. Fig 5 illustrates the connection between several IoT devices with A71CH IC and the AWS IoT cloud.

The credentials of each one of the involved elements are explained in this chapter. Additionally, basic concepts on SSL/TLS and security are presented to provide the reader the necessary background for a better understanding of the presented contents such as TLS handshake protocol, OpenSSL and the A71CH OpenSSL engine.

Fig 5. System overview. Connection between an IoT device(s) and AWS IoT cloud

4.1 OEM and AWS cloud credentials

The IoT device original equipment manufacturer (OEM) shall have an intermediate CA certificate, as well as their corresponding keys. This intermediate CA certificate will be issued by a root CA. Regarding the root CA there are two possibilities. The OEM can have its own public key infrastructure (PKI); thus, it will have the root CA certificate and the root CA key pair. Alternatively, the OEM can trust in a third-party CA.

Note that in Fig 6 only an intermediate CA and a root CA have been represented, though the OEM could have more than one intermediate CA issued by different root CAs.

The OEM will have to register the intermediate CA certificate in AWS cloud. The intermediate CA will be used to validate all the IoT devices certificates willing to connect to the cloud.
Each IoT device will be provisioned with its corresponding key pair and its digital certificate issued by the intermediate CA. These credentials will be stored inside the A71CH and will be used to register the IoT device certificate and establish a secure connection with the AWS IoT cloud.

The IoT device must also contain the intermediate CA certificate to connect to AWS. Optionally, it could be stored inside the A71CH GP Storage for functionality purposes. Fig 7 illustrates the credentials of an IoT device.

Fig 8 shows the complete connection between IoT devices and AWS cloud. The credentials described in 4.1 and 4.2 have also been represented.
Connection between IoT devices and AWS IoT cloud

**Fig 8.** Connection between IoT devices and AWS IoT cloud

### 4.3 Transport Layer Security protocol (TLS)

IoT devices own several connectivity features that allow them to exchange data with the cloud. The network link between these IoT devices and the cloud or server should be secure. Transport Layer Security protocol (TLS), and its predecessor Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over unsecure networks. These protocols are created from the necessity of establishing a connection preserving confidentiality, integrity and authenticity.

**Fig 9.** TLS connection between two IoT devices and AWS IoT cloud

Fig 10 illustrates the protocol stack of a TLS communication over a TCP/IP network. In the well-known ISO/OSI layer architecture, SSL/TLS would belong to the Presentation Layer in charge of encrypting and securing the entire communication. The transport and network protocol TCP/IP and the medium access control (MAC) would fall in layers from 4 to 2, respectively. Finally, data would be physically transferred according to ethernet (or wireless ethernet) protocols.
4.3.1 Transport Layer Security Handshake protocol

Before the IoT device and the server in the cloud begin exchanging data over TLS, the tunnel encryption must be negotiated. This negotiation is referred as TLS Handshake. The TLS Handshake Protocol is responsible for the authentication and key exchange necessary to establish or resume secure sessions. When establishing a secure session, the TLS Handshake Protocol manages the following:

- Agree on the TLS protocol version to be used.
- Select cipher suite.
- Authenticate each other by exchanging and validating digital certificates.
- Use asymmetric encryption techniques to generate a shared secret key, which avoids the key distribution problem. SSL or TLS then uses the shared key for the symmetric encryption of messages, which is faster than asymmetric encryption.

The TLS Handshake Protocol involves the following steps:

- Exchange Hello messages to agree on algorithms, exchange random values, and check for resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a pre-master secret.
- Exchange certifications and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the pre-master secret and exchanged random value.
- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.
The A71CH security IC supports the TLS Handshake Protocol version 1.2 with the following options:

- Pre-Shared Key Cipher suites for TLS as described in [RFC4279]: A set of cipher suites for supporting TLS using pre-shared symmetric keys (TLS_PSK_WITH_xxx)
- ECDHE_PSK Cipher suites for TLS as described in [RFC5489]: A set of cipher suites that use a pre-shared key to authenticate an Elliptic Curve Diffie-Hellman exchange with Ephemeral keys (TLS_ECDHE_PSK_WITH_xxx).

The Fig 11 represents an overview of the TLS 1.2 handshake with ECDSA-ECDHE. More information about the TLS 1.2 handshake protocol can be obtained from the standard specifications document [RFC5246] or from [AN_A71CH_HOST_SW].
Fig 11. TLS 1.2 Handshake diagram with ECC
4.3.2 Transport Layer Security software libraries

There are several full-featured TLS software libraries that can be used in both server cloud and IoT devices such OpenSSL, mbedTLS, WolfTLS, etc.

OpenSSL [OPEN_SSL] is an open-source implementation of SSL/TLS protocol. It is written in C language, although there are several wrappers to use this library in other languages. It implements all the cryptography functions needed and it is widely used. Starting with OpenSSL 0.9.6, an ‘Engine interface’ was added allowing support for alternative cryptographic implementations. This Engine interface can be used to interface with external crypto devices as e.g. HW accelerator cards or security ICs like the A71CH.

The OpenSSL toolkit including an A71CH OpenSSL Engine is available as part of the A71CH Host software package [A71CH_OPENSSL_ENGINE]. The A71CH OpenSSL Engine gives access to several A71CH features via the A71CH Host Library not natively supported by OpenSSL implementation. In other words, the Engine links the OpenSSL libraries to the A71CH Host API and overwrites some of the native OpenSSL functions in order to include the use of the A71CH crypto functionality such as sign, verify and key exchange operations or random messages generation, that can be used for instance during the TLS Handshake protocol.

The A7CH OpenSSL Engine is fully compatible with the i.MX6UltraLite embedded platform. Nevertheless, support will be added in future revisions.

Fig 12 illustrates the IoT Host MCU software architecture. As it can be observed, the software stack is formed by an application that calls OpenSSL functions. Some of these functions will be overwritten by the A71CH OpenSSL Engine, thus the A71CH crypto functionality will be used through the A71CH Host Library over I²C.
5. Introduction to AWS IoT and JITR

AWS has built IoT specific services. These services help IoT system developer collect and send IoT device data to the cloud, make it easy to load and analyze that information, and provide the ability to manage connected IoT devices.

AWS IoT Core service is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices. IoT Core can support billions of devices and trillions of messages and can process and route those messages.

Fig 13 illustrates the end-to-end flow to establish a secure connection between an IoT device with an A71CH to the AWS IoT services.

As a first step, the A71CH ICs provided by NXP need to be provisioned by a Programming facility in charge of injecting die-individual credentials (presented in Chapter 4). After that, the OEM manufacturer needs to register and activate his intermediate CA in the AWS platform. Then, in order to communicate in the AWS IoT network, each IoT device needs to be known via its device certificate. Thus, each IoT device certificate needs to be registered into the AWS IoT platform.

---

**Fig 13. End-to-end connection establishment flow**

In order to ease this registration process, AWS introduced the Just-In-Time-Registration (JITR) mechanism. The AWS JITR is a mechanism to **automatically** register new device certificates during the initial communication between the IoT device and the AWS IoT cloud. Using AWS JITR mechanism, an IoT device can be connected to AWS IoT cloud if the OEM follows these two steps:

- Set up the AWS IoT account by registering and activating the intermediate CA certificate.
- Configure the AWS IoT account for the first-time device onboarding. Then, connect the IoT device to AWS IoT cloud to register its digital certificate automatically.

These two steps are further elaborated in Chapter 6 and Chapter 7. In addition, more information on AWS IoT services and the Just-In-Time registration mechanism can be found in [AWS_IOT] [JITR].
6. **AWS setup by OEM: Register intermediate CA certificates**

The OEM manufacturer should start setting up the AWS account by registering the intermediate CA digital certificate. This certificate must be uploaded to the AWS to enable auto-registration of certificate via JITR. Every time a new IoT device is connected to the AWS cloud, the chain of trust will be validated, i.e., the IoT device certificate signature is validated by the intermediate CA certificate, and at the same time, the intermediate CA certificate signature must have been validated by the root CA.

The workflow to register and activate a new intermediate CA certificate in AWS cloud is the following:

- Obtain a registration code from AWS.
- Use the registration code to create a Verification CSR. The registration code will be embedded in the 'CN' (Common Name) field of the Verification CSR.
- Use the CSR and the intermediate CA credentials to create a Verification Certificate.
- Upload both the Verification certificate and the intermediate CA certificate to AWS.
- Activate the registered intermediate CA certificate, after validation of the intermediate signature.
- Enable auto-registration status of the registered intermediate CA certificate.

These required steps are described in this chapter. The entire workflow can be done using either the AWS CLI (command line interface) or the register certificate section of the AWS IoT website. In this document though, the AWS CLI has been used.

### 6.1 AWS Command Line Interface (AWS CLI)

The AWS Command Line Interface (CLI) is an open source tool built on top of the AWS SDK for Python (Boto) that provides commands for interacting with AWS services and direct access to AWS service’s public APIs. It requires minimal configuration and once installed it can be used from a terminal application in Windows, Linux shells or even through a remote terminal such as PuTTY or SSH on Amazon EC2 instances. More information on AWS CLI can be found in [AWS_CLI](#).

The AWS CLI can be downloaded from [AWS_CLI](#) and installed on Windows, Mac and Linux platforms. Fig 14 shows a capture of the execution of AWS CLI with Windows PowerShell. The ‘help’ command has been executed to prompt all the AWS CLI functionality on the terminal.
6.2 Obtain registration code from AWS

The process for validating ownership and registering an intermediate CA certificate in AWS IoT services is done through a challenge and response flow.

To register an intermediate CA certificate with AWS IoT, OEM have to verify that you have access to both the intermediate CA certificate and the intermediate CA private key. For this, a Verification certificate has to be generated using a registration code provided by AWS IoT and the CA private key.

The registration code can be obtained with the following AWS CLI command:

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

The provided code is randomly generated and long-lived, i.e., it will not expire.

6.3 Create Verification certificate and upload it to AWS

Then, a certificate signing request (CSR) has to be prepared to generate the Verification certificate. This CSR will contain the registration code in its 'CN' field (highlighted in green). The following OpenSSL commands can be executed:

```bash
openssl ecparam -genkey -name prime256v1 -out VerificationKeys.pem
openssl req -new -key VerificationKeys.pem -subj "/CN=registration code" -out VerificationCSR.pem
openssl x509 -req -in VerificationCSR.pem -CA intCACertificate.pem -CAkey intCAkey.pem -CAcreateserial -out VerificationCertificate.pem -days 365 -sha256
```

Firstly, a new ECC key pair is generated. This key pair will belong to the Verification certificate. Then the CSR containing the registration code is created and the Verification certificate is created based upon the information contained in the CSR and signed with the intermediate CA private key. Finally, both the Verification certificate and the intermediate CA certificate are uploaded to AWS IoT using the AWS CLI:

```bash
$aws iot register-ca-certificate --ca-certificate file://intCA_certificate.pem --verification-certificate file://VerificationCertificate.pem
```
6.4 Activate intermediate CA certificate and enable auto-registration

By default, the registered intermediate CA certificate is in ‘inactive’ state, which means that it cannot be used for validating new IoT devices certificates. Hence, it is necessary to activate it.

The information of the registered certificate can be obtained with the following command:

```bash
$ aws iot describe-ca-certificate --certificate-id <certificateId>
```

Where `<certificateId>` is the ID returned in the response of the previous AWS CLI command (`iot register-ca-certificate`).

Then, the intermediate CA can be activated:

```bash
$ aws iot update-ca-certificate --certificate-id <certificateId> --new-status ACTIVE
```

Also, the auto-registration-status property of the registered intermediate CA certificate has to be enabled. If auto-registration-status is enabled, AWS IoT services will automatically register any IoT device certificate issued by that intermediate CA.

```bash
$ aws iot update-ca-certificate --certificate-id <certificateId> --new-auto-registration-status ENABLED
```

Fig 15 illustrates the explained CA certificate registration workflow.
**Fig 15. CA certificate registration workflow**

1. **Fetch Registration Code**
   - AWS

2. **Obtain Registration Code**
   - Reg. code

3. **Create Verification certificate key pair**
   - OPENSSL

4. **Prepare Verification CSR**
   - OPENSSL

5. **Upload Verification certificate and CA certificate**
   - AWS

6. **Register intermediate CA certificate**
   - AWS

7. **Activate the registered CA certificate**
   - AWS

8. **Enable auto registration status property**
   - AWS

9. **Update certificate ‘ACTIVE’ property**

10. **Update certificate ‘auto-register’ property**

**Verification key pair**

- PRIV.
- PUB.

**Intermediate CA key pair**

- PRIV.
- PUB.

**VERIFICATION CERT.**

**INTERMEDIATE CA CERT.**
6.5 Register CA certificates with AWS IoT website interface

The intermediate CA registration process can also be carried out through the user interface available in the AWS IoT website. Fig 16 shows a screen capture of the CA certificate registry section. As can be observed, the same steps have to be followed, i.e., obtain a registration code, prepare the Verification CSR with the registration code in the "Common Name" field, sign the Verification CSR with the intermediate CA and upload it together with the intermediate CA.

![Register a CA certificate](image)

To use your own X.509 certificates, you must register a CA certificate with AWS IoT. You must prove own the private key associated with the CA certificate by creating a private key verification certificate. The CA certificate can then be used to sign device certificates. You can register up to 10 CA certificates with the same subject field and public key per AWS account. This allows you to have more than one CA sign your device certificates.

**Step 1: Generate a key pair for the private key verification certificate**

```
openssl genrsa -out verificationCert.key 2048
```

**Step 2: Copy this registration code**

```
66eb35e359f1f53a5e5421b4b48436764c17e99bdf791119abda5a90a93a7b
```

**Step 3: Create a CSR with this registration code**

```
openssl req -new -key verificationCert.key -out verificationCert.csr
```

Put the registration code in the **Common Name** field

<table>
<thead>
<tr>
<th>Country Name (2 letter code) [AU]:</th>
<th>State or Province Name [full name] [Some-State]:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Localitly Name [eg. city]:</td>
<td>Organization Name [eg. company] [Internet Widgets Pty Ltd]:</td>
</tr>
<tr>
<td>Organizational Unit Name [eg. section]:</td>
<td>Common Name [eg. server FQDN or YOUR name]: 606eb35e339f953a5e5421b4b48436764c17e99bdf791119abda5a90a93a7b</td>
</tr>
<tr>
<td>Email Address []:</td>
<td></td>
</tr>
</tbody>
</table>

**Step 4: Use the CSR that was signed with the CA private key to create a private key verification certificate**

```
openssl x509 -req -in verificationCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out verificationCert.crt -days 3600 -sha256
```

**Step 5: Upload the CA certificate (rootCA.pem)**

**Select CA certificate**

**Step 6: Upload the verification certificate (verificationCert.crt)**

**Select verification certificate**

- [ ] Activate CA certificate
- [ ] Enable auto-registration of device certificates

---

Fig 16. Register CA certificate AWS IoT website interface
7. First time device onboarding: IoT device certificate registration

The intermediate CA certificate is already registered, activated and its ‘auto-registration-status’ has been enabled. Every time a new IoT device is connected to the AWS IoT cloud, the service will detect an unidentified IoT device certificate issued by that intermediate CA and will proceed to auto-register it. The first-time device onboarding workflow can be divided into the following two steps:

- Combine the IoT device certificate with the intermediate CA certificate.
- Initiate a TLS communication with AWS IoT cloud using an MQTT client.

First of all, the IoT device will have to prepare a file composed of the IoT device certificate and the intermediate CA certificate. Then, the IoT device will attempt to establish a TLS connection with AWS IoT using an MQTT client. The AWS IoT service supports the publish-subscribe MQTT.

During the TLS handshake, AWS IoT will detect that the IoT device has not been registered before and therefore will proceed to register it on AWS IoT automatically. The registered IoT device certificate will be ‘de-activated’ by default. To automatically activate a new IoT device, an AWS Lambda function has to be previously created and properly configured.

Finally, the IoT device will be able to establish a TLS connection with AWS IoT.

All these steps are explained in detail in this chapter, as well as the MQTT protocol and the AWS Lambda functions.

7.1 Combine IoT device certificate with intermediate CA certificate

The IoT device certificate has to be combined with the issuer intermediate CA certificate. In Linux based platforms, it can be done with ‘cat’ terminal command:

```
$ cat IoTdeviceCertificate.pem intCAcertificate.pem > CAandIoTcertificate.pem
```

While in Windows-based environments, it can be done by running the following command on the shell:

```
type IoTdeviceCertificate.pem intCAcertificate.pem > CAandIoTcertificate.pem
```

An MQTT client running across TLS will be used to establish a TLS link between the IoT device and AWS IoT. During the TLS handshake, the “CAandIoTcertificate” file will be sent to AWS IoT.

7.2 MQTT

Message Queue Telemetry Transport (MQTT) is an ISO standard (ISO/IEC PRF 20922) Machine-to-Machine (M2M) publish-subscribe based communication protocol used in the Internet of Things. The MQTT is a low-power and low-bandwidth protocol oriented to embedded devices with small computational resources such as sensors. For this reason, it is commonly employed in sensor networks, i.e., IoT devices networks.

MQTT takes layers 5-7 of the ISO/OSI layers model and relies on TCP/IP as transport and network protocols. Additionally, it supports the use of SSL/TLS for encrypting and
securing the communication. Fig 17 illustrates the communication stack presented in 4.3, including MQTT.

![Fig 17. Communication stack with MQTT and TLS. ISO/OSI Layers](image)

Regarding the network architecture, 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 are periodically transmitting messages and waiting for the broker response. Fig 18 illustrates an example of an MQTT star architecture. In this case, an AWS IoT cloud acts as the broker device and a series of IoT devices are the clients.

![Fig 18. Example of MQTT star architecture](image)

Communication between client devices and the broker is based on ‘topics’. A topic is a UTF-8 string which is used by the broker to filter messages for each connected client.
topic can consist of one or more levels, each one separated by a forward slash. Clients can publish messages in a given topic, or they can subscribe to a topic to receive messages published by the rest of the client devices.

For instance, in the AWS IoT scenario, the following MQTT topic is used by the broker (AWS IoT) for managing the registered IoT device certificates (clients):

```
$aws/events/certificates/registered/<caCertificateID>
```

Whenever an IoT device wants to establish a TLS connection with AWS IoT, the IoT device certificate will be first checked. First, the signature of the IoT device certificate will be validated with the public key contained in the registered intermediate CA certificate.

In case the IoT device certificate signature is validated, but the certificate is not registered in AWS IoT (or it is in ‘PENDING_ACTIVATION’ state), the TLS handshake will fail.

Then, the IoT device certificate will be auto-registered in the AWS IoT server and an MQTT message will be automatically published by AWS IoT in that ‘registration’ topic. Remember that, the ‘auto-registration’ property of the registered intermediate CA should be enabled as it is explained in 6.4.

The published MQTT message has the following structure:

```json
{
    "certificateId": "<certificateID>",
    "caCertificateId": "<caCertificateId>",
    "timestamp": "<timestamp>",
    "certificateStatus": "PENDING_ACTIVATION",
    "awsAccountId": "<awsAccountId>",
    "certificateRegistrationTimestamp": "<certificateRegistrationTimestamp>",
}
```

Fig 19. Example of MQTT certificate registration publication

where `<caCertificateID>` belongs to the identifier of the intermediate CA certificate. Furthermore, as it can be observed, the `certificateStatus` field is in ‘PENDING_ACTIVATION’.

More information on MQTT and MQTT with AWS can be found in [MQTT] [JITR].

### 7.3 IoT device certificate registration event with AWS Lambda function

As has been mentioned in 7.2, when AWS IoT publishes an MQTT ‘certificate registration’ message, the registered IoT device certificate `certificateStatus` field is in ‘PENDING_ACTIVATION’ state.

Further TLS connection attempts will continue failing because only ‘ACTIVE’ certificates are authenticated with AWS IoT. Therefore, the `certificateStatus` field of the registered IoT device should be changed to ‘ACTIVE’ in order to be successfully authenticated
during the TLS handshake. It is possible to automatically move certificate status from ‘PENDING_ACTIVATION’ to ‘ACTIVE’ by using AWS Lambda functions.

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. For instance, it is possible to attach a rule to an AWS IoT MQTT topic that can take some action based on the messages received.

Therefore, regarding the IoT device certificates registration, it is possible to create a rule for the ‘registration’ topic that automatically activates the registered IoT device certificates. A step-by-step guide explaining how to create a new AWS Lambda function and attach a new rule to a MQTT topic is presented in detail in [JITR].

Also, more information on AWS Lambda functionality can be found in [AWS_LAMBDA].

7.4 Connection to AWS IoT with an MQTT client

Finally, the IoT device will attempt to establish a TLS connection with AWS IoT. In this example, the MQTT Mosquitto client is used [MQTT_MOSQUITTO], and the following command is executed:

```
$ mosquitto_pub --cafile root.cert --cert CAandIoTcertificate.pem --key IoTdeviceKey.pem -h <prefix>.iot.us-east-1.amazonaws.com -p 8883 -q 1 -t foo/bar -i anyclientID --tls-version tlsv1.2 -m "Hello" -d
```

where:
- “root.cert” (–ca file) is the AWS IoT root certificate used by an IoT device to verify the identity of the AWS IoT servers during the TLS Handshake (4.3.1). This AWS root certificate can be downloaded from [AWS_ROOT].
- “CAandIoTcertificate.pem” (–cert) is the file containing the IoT device certificate and the intermediate CA certificate.
- “IoTdeviceKey.pem” (–key) is the IoT device key pair.
- “<prefix>. iot.us-east-1.amazonaws.com” (-h) is the AWS IoT service instance.
- “8883” (-p) is the port number.
- “1” (-q) specifies the quality of service to use for the message.
- “foo/bar” (-t) specifies the topic on which to publish the message.
- “anyclientID” (-i) specifies the id to use for this client.
- “tlsv1.2” (–tls-version) specifies which TLS protocol version to use when communicating with AWS.
- “Hello” (-m) is the message to be sent.
- Finally (-d) enables debug messages.

This command will lead to the auto-registration of the IoT device certificate during the TLS handshake:
- The TLS Handshake protocol will start.
- A registration message (Fig 19) will be published by AWS in the MQTT ‘registration’ topic in order to register the IoT device certificate.
• The TLS Handshake will fail since AWS IoT disconnects the connection after the registration of the IoT device certificate.

• The attached AWS Lambda rule will be triggered and the ‘certificateStatus’ field of the publication will be automatically changed from ‘PENDING_ACTIVATION’ to ‘ACTIVE’.

• The IoT device could implement an automatic reconnect strategy to correctly establish a TLS connection with AWS IoT cloud.

Fig 20 illustrates the explained IoT device certificate registration workflow.

---

**Note:** This example is just a conceptual illustration on how the MQTT client works. In this example the A71CH is not involved in the TLS handshake protocol. The next chapter takes the reader step-by-step on how to achieve the same using an A71CH.
Fig 20. IoT device certificate registration workflow
8. Evaluating A71CH for secure connection to AWS IoT

This section concerns how to establish a secure connection between an IoT device featuring the A71CH and AWS IoT cloud. In this example, the following elements are involved:

- 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 MCIMX6UL-EVK.
- Development PC: a Windows platform will be used to configure and prepare the AWS IoT account and register a demo CA certificate. Additionally, the i.MX6UltraLite will be controlled from the development PC using Tera Term.

A quick-start guide on how to get started with the OM3710/A71CHARD development kit and the MCIMX6UL-EVK i.MX6UltraLite evaluation board can be found in [QUICK_START_IMX6]. Regarding the i.MX6UltraLite Internet connection, it might be necessary to manually set the IP address of the DNS server.

**Note:** This section describes how to establish a secure connection with AWS using a development PC and an i.MX6UltraLite with an A71CH security IC. The following description will re-uses several of the commands already presented in this document and it is provided only for demonstration. Therefore, the subsequent procedure must be adapted and adjusted accordingly for commercial deployment.

![Fig 21. Demo system setup](image)

Fig 21 depicts the setup that will be used for this demonstration. The connection with AWS IoT cloud will be established from the i.MX6UltraLite MCU. The A71CH security IC will be connected to the i.MX6UltraLite through the OM3710/A71CHARD Arduino shield.

First, the user acting as the OEM will use a development PC to register and activate an intermediate CA certificate as it is explained in chapter 6. In this example, it is assumed that there is no existing intermediate CA; thus, a self-signed demo CA certificate will be created only for demo purposes.
Windows OpenSSL libraries (included in the A71CH Host software package) will be used to create the demo CA credentials, while AWS CLI will be used to configure OEM’s AWS account and to register the demo CA certificate. Both OpenSSL and AWS CLI commands will be called from a Windows command prompt ‘cmd’. Once the demo CA certificate is registered and activated, the demo CA credentials will be transferred to the i.MX6UltraLite.

In the i.MX6UltraLite, the demo CA certificate will be combined with the IoT device certificate and the connection with AWS will be established as explained in chapter 7. For this purpose, the IoT device credentials will be created in the i.MX6UltraLite using OpenSSL libraries, and the IoT device key pair will be securely stored into the A71CH using the Configure tool. Finally, the TLS connection with AWS IoT will be established using an MQTT client compatible with the A71CH OpenSSL engine.

To summarize, the steps are the following:

1. Configure the AWS IoT account and obtain a registration code using AWS CLI.
2. Using OpenSSL, create demo CA credentials and the Verification certificate with the registration code.
3. Upload both the Verification certificate and the demo CA certificate to AWS IoT using the AWS CLI.
4. Activate the registered demo CA and enable ‘auto-registration’ status using AWS CLI.
5. Transfer the demo CA credentials from the development PC to the i.MX6.
6. Using OpenSSL on the i.MX6UltraLite, prepare the IoT device credentials and combine the IoT device certificate with the demo CA certificate.
7. Connect to AWS IoT using an MQTT client.

In this chapter, each one of the listed steps will be explained.

### 8.1 Obtain registration code from AWS

The first step is to obtain a registration code from AWS IoT. For this, the AWS CLI will be used as stated in section 6.2. The AWS CLI will be launched from a Windows command line terminal, and the following command will be called to first configure the AWS IoT account:

```
aws configure
```

AWS Access credentials (Access Key ID and Secret Access Key), region name and output format will be asked. The AWS Access Key ID and Secret Access Key can be obtained from the AWS IoT control panel console [AWS_CLI_ACCESS], while the region name can be obtained from [AWS_CLI_REGIONS]. Finally, the default output format will be set as ‘text’ (by default, it is configured as ‘json’).

Once the AWS CLI account has been configured, a registration code can be requested:

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

Fig 22 shows the command line terminal and the output of both ‘configure’ and ‘iot get-registration-code’ commands. The AWS Access Key ID, AWS Secret Access Key and registration code have been blurred.
8.2 Create Verification certificate and upload it to AWS

The demo CA credentials and the Verification certificate will be created in the Windows platform using OpenSSL commands (section 6.3) in a command line terminal. The following steps have to be taken:

- First, a new pair of ECC keys `CACertificate_ECC.key` will be created for the demo CA credentials.
- Then, a self-signed `CACertificate_ECC.crt` certificate will be created.
- The Verification ECC key pair `VerificationKeys.key` will be generated.
- The Verification public key and the registration code obtained in 0 will be included in the `VerificationCSR.pem`.
- The Verification certificate `Verificationcertificate.crt` will be issued by the demo CA certificate `CACertificate_ECC.crt`. It will contain the information retrieved from the `VerificationCSR.pem`.

Each one of the listed steps corresponds to the following OpenSSL commands:

```bash
openssl ecparam -genkey -name prime256v1 -out CACertificate_ECC.key
openssl req -x509 -new -nodes -key CACertificate_ECC.key -sha256  -days 3650 -out CACertificate_ECC.crt -subj "/CN="Demo CA NXP"
openssl ecparam -genkey -name prime256v1 -out VerificationKeys.key
openssl req -new -key VerificationKeys.key -subj "/CN=REGISTRATION CODE" -out VerificationCSR.pem
openssl x509 -req -in VerificationCSR.pem -CA CACertificate_ECC.crt -CAkey CACertificate_ECC.key -CAcreateserial -out Verificationcertificate.crt -days 3650 -sha256
```

Once the Verification certificate has been created, it will be registered to AWS IoT using AWS CLI with the `iot register-ca-certificate` command:

```
aws configure
AWS Access Key ID [***************H0N0]:
AWS Secret Access Key [***************vP6V]:
Default region name [eu-central-1]:
Default output format [text]:
aws iot get-registration-code
```

Fig 22. AWS CLI configure command

Fig 23. OpenSSL commands executed in the development PC

Fig 23 shows the Windows terminal with the execution of the above-mentioned commands. The registered demo CA certificate information can be prompted with the following AWS command (section 6.4):

aws iot describe-ca-certificate --certificate-id 8c2c1cde98712d88a7daed538bdfde3b4685f7534f695b7f260ae99cd08d57d

Where the certificate ID ‘8c2c1cde…’ is the string returned in the response of the ‘iot register-ca-certificate’ command. As is highlighted in Fig 24, the registered CA certificate is set as inactive, and its auto-registration property is disabled by default.

Fig 24. Registered demo CA

It is also possible to observe the status of the registered CA certificate directly on the AWS IoT website interface (Fig 25).
Activate demo CA certificate and enable auto-registration

The registered CA certificate can be activated, and its 'auto-registration' property can be enabled with the following AWS CLI commands:

```bash
aws iot update-ca-certificate --certificate-id 8c2c1cde98712d88a7daed538bdfde3b4685f7534f695b7f260ae99cd08d57d --new-status ACTIVE
aws iot update-ca-certificate --certificate-id 8c2c1cde98712d88a7daed538bdfde3b4685f7534f695b7f260ae99cd08d57d --new-auto-registration-status ENABLE
aws iot describe-ca-certificate --certificate-id 8c2c1cde98712d88a7daed538bdfde3b4685f7534f695b7f260ae99cd08d57d
```

Fig 26 shows the final state of the registered CA. The 'auto-registration' property has been enabled, and the certificate has been activated.
8.4 Transfer demo CA credentials to the i.MX6 MCU

The demo CA credentials have to be transferred to the i.MX6UltraLite, for instance using SCP (Secure Copy Protocol). Using SCP permits the OEM to distribute the CA certificates remotely to all their IoT devices connected to the same network in a secure way.

Fig 27 shows the ‘pscp -scp’ command executed from the Windows command prompt. It shows the transfer of both demo CA credentials from the development PC to the i.MX6UltraLite platform. This ‘pscp -scp’ command is provided by installing the PuTTY client [PUTTY].

SCP and OpenSSL tools can also be easily obtained by installing Cygwin on a Windows PC [CYGWIN].

8.5 Create IoT device credentials and combine IoT device certificate with the demo CA certificate

The i.MX6UltraLite will be controlled from the development PC using the Tera Term terminal as is explained in [QUICK_START_IMX6]. For the sake of simplicity, the IoT device credentials will be created in the same folder where the A71CH Configure tool executable and the demo CA credentials are located. The following OpenSSL commands can be used:

```bash
openssl ecparam -genkey -name prime256v1 -out deviceKey.key
```
Fig 28 shows the Tera Term window with the OpenSSL commands executed from the i.MX6UltraLite.

```
openssl req -new -key deviceKey.key -out deviceCsr.csr -subj "/CN=IoT device"
openssl x509 -req -days 3650 -in deviceCsr.csr -CAcreateserial -CA CACertificate_ECC.crt -CAkey CACertificate_ECC.key -out deviceCert.crt
```

Finally, the IoT device certificate `deviceCert.crt` is combined with the demo CA credentials using the ‘cat’ command as explained in 7.1:

```
cat deviceCert.crt CACertificate_ECC.crt > CAandIoTcert.pem
```

Fig 29 shows the created IoT device credentials and the file composed of the IoT device certificate and the demo CA certificate in the i.MX6UltraLite file system.

The IoT device keys can be stored in the A71CH using the A71CH Configure tool and the following commands:

```
./a71chConfig_i2c_imx debug reset
./a71chConfig_i2c_imx set pair -x 0 -k deviceKey.key
./a71chConfig_i2c_imx info pair
./a71chConfig_i2c_imx refpem -c 10 -x 0 -r deviceRefKey.ref_key
```

Where `deviceRefKey.ref_key` is the reference to the stored key. This reference file will be further employed by the i.MX6 to use the IoT device key stored inside the A71CH IC.
Fig 30. Injection of credentials with A71CH Configure tool

Fig 30 shows the Tera Term screen with the execution of the A71CH Configure tool commands. Each one of the commands has been highlighted. Also, the stored key and the created reference file `deviceRefKey.ref_key` have been highlighted.

### 8.6 Connection to AWS IoT with an MQTT client

The last step consists of establishing a TLS connection with AWS IoT.

This can be done by using the available AWS IoT SDK [AWS_IOT_SDK].
## 9. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A71CH_ANTICOUNTERFEIT]</td>
<td>AN12120 A71CH for electronic anti-counterfeit – Application note, document number 4583**1</td>
</tr>
<tr>
<td>[RFC4279]</td>
<td>Pre-Shared Key Ciphersuites for Transport Layer Security (TLS) - December 2005</td>
</tr>
<tr>
<td>[A71CH_OPENSSL_ENGINE]</td>
<td>A71CH OpenSSL Engine – DocStore, um4334**1</td>
</tr>
<tr>
<td>[AN_A71CH_HOST_SW]</td>
<td>AN12133 A71CH Host software package documentation – Application note, document number 4643**1</td>
</tr>
<tr>
<td>[QUICK_START_IMX6]</td>
<td>AN12119 Quick start guide for OM3710A71CHARD i.MX6 – Application note, document number 4582**1</td>
</tr>
<tr>
<td>[PUTTY]</td>
<td>PuTTY client - <a href="https://www.putty.org/">https://www.putty.org/</a></td>
</tr>
</tbody>
</table>

* **… document version number*
10. Legal information

10.1 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.

10.2 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. NXP Semiconductors products are not designed, authorized or warranted to be used in life support, lift-critical or safety-critical systems or equipment. NXP Semiconductors state that these products do not infringe any patents of any third party. NXP Semiconductors reserves the right to make changes, without notice, to its products or specifications at any time and to discontinue at any time any NXP Semiconductors products or services. NXP Semiconductors disclaims all warranties, whether express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. NXP Semiconductors shall not be liable to the customer for any special, indirect, consequential, punitive or incidental damages or costs of any nature (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 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 the 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.

10.1 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.

10.2 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.

PC-bus — logo is a trademark of NXP B.V.
11. List of figures

Fig 1. Digital signature diagram ........................................4
Fig 2. Digital certificate generation steps and contents ..........................................................5
Fig 3. Elliptic Curve Digital Signature Algorithm (ECDSA) example..................................5
Fig 4. Elliptic Curve Diffie-Hellman Key Exchange (ECDH) example.................................6
Fig 5. System overview. Connection between an IoT device(s) and AWS IoT cloud ............7
Fig 6. OEM and AWS cloud credentials .................................................................8
Fig 7. IoT device credentials .................................................................8
Fig 8. Connection between IoT devices and AWS IoT cloud ..................................................9
Fig 9. TLS connection between two IoT devices and AWS IoT cloud ........................................9
Fig 10. Communication stack. ISO/OSI Layers.........................10
Fig 11. TLS 1.2 Handshake diagram with ECC ..........................12
Fig 12. Host SW stack including OpenSSL, A71CH OpenSSL engine and A71CH Host Library .......................13
Fig 13. End-to-end connection establishment flow ..................14
Fig 14. AWS CLI on Windows PowerShell ..........................16
Fig 15. CA certificate registration workflow ..................18
Fig 16. Register CA certificate AWS IoT website interface ........................................19
Fig 17. Communication stack with MQTT and TLS. ISO/OSI Layers.................................21
Fig 18. Example of MQTT star architecture ..................21
Fig 19. Example of MQTT certificate registration publication ........................................22
Fig 20. IoT device certificate registration workflow ..................25
Fig 21. Demo system setup ........................................26
Fig 22. AWS CLI configure command ..................28
Fig 23. OpenSSL commands executed in the development PC ........................................29
Fig 24. Registered demo CA ................................................29
Fig 25. AWS IoT website interface ........................................30
Fig 26. Registered demo CA ................................................31
Fig 27. Transfer the demo CA credentials using SCP ..................31
Fig 28. OpenSSL commands executed in the i.MX6 MCU ........................................32
Fig 29. Created IoT device credentials ..................................32
Fig 30. Injection of credentials with A71CH Configure tool ..................................33
12. List of tables

Table 1. Referenced Documents .................................. 34
13. Contents

1. Introduction ......................................................... 3
2. A71CH overview .................................................. 3
3. Public key infrastructure and ECC fundamentals .......................... 3
   3.1 Digital signature ........................................... 4
   3.2 Digital certificate, Certification Authority (CA) and Certificate Signing Request (CSR) .... 4
   3.3 Elliptic Curve Digital Signature Algorithm (ECDSA) ........................................ 5
   3.4 Elliptic Curve Diffie-Hellman (ECDH) ........................................... 5
   3.5 A71CH ECC supported functionality .................................. 6
4. A71CH for secure connection to AWS IoT cloud .................................. 7
   4.1 OEM and AWS cloud credentials ....................................... 7
   4.2 IoT device credentials ........................................... 8
   4.3 Transport Layer Security protocol (TLS) ..................................... 9
   4.3.1 Transport Layer Security Handshake protocol ...................... 10
   4.3.2 Transport Layer Security software libraries ................... 13
5. Introduction to AWS IoT and JITR .................................. 14
6. AWS setup by OEM: Register intermediate CA certificates ................. 15
   6.1 AWS Command Line Interface (AWS CLI) .......... 15
   6.2 Obtain registration code from AWS ....................... 16
   6.3 Create Verification certificate and upload it to AWS ....................... 16
   6.4 Activate intermediate CA certificate and enable auto-registration ................. 17
   6.5 Register CA certificates with AWS IoT website interface .............. 19
7. First time device onboarding: IoT device certificate registration ......... 20
   7.1 Combine IoT device certificate with intermediate CA certificate .................. 20
   7.2 MQTT ........................................................... 20
8. Evaluating A71CH for secure connection to AWS IoT ................................ 26
   8.1 Obtain registration code from AWS ..................... 27
   8.2 Create Verification certificate and upload it to AWS ....................... 28
   8.3 Activate demo CA certificate and enable auto-registration ................. 30
   8.4 Transfer demo CA credentials to the i.MX6 MCU ................. 31
   8.5 Create IoT device credentials and combine IoT device certificate with the demo CA certificate .... 31
   8.6 Connection to AWS IoT with an MQTT client .... 33
9. References ................................................................ 34
10. Legal information ................................................ 35
    10.1 Definitions .................................................. 35
    10.2 Disclaimers ............................................... 35
    10.3 Licenses ................................................... 35
    10.4 Trademarks ............................................... 35
11. List of figures .................................................. 36
12. List of tables ................................................... 37
13. Contents .......................................................... 38

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

© NXP Semiconductors N.V. 2018. All rights reserved.

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

Date of release: 29 March 2018
Document identifier: 464110