SHC RISC-V RoT Demo

Andes RISC-V SoC with FreeRTOS and Amazon IoT Core

April 22, 2021
SH Consulting Group

SHC Demonstration: Andes Corvette F1 RoT with AWS IoT Core

This demo shows a Secure IoT solution using a Corvette-F1 board, evaluation kit with full support for the 32-bit AndesCore N25 and the AndeShape AE250 Platform, runs Amazon FreeRTOS, which is an open source operating system for microcontrollers from Amazon Web Services (AWS). It uses an ESP32-WROOM board as an external Wi-fi module. ATECC608A-MAHDA chip is integrated such as Trusted Platform Module (TPM) to provide hardware-based endpoint device security. This integration ensures the private key used to establish device identity can be securely stored in tamper-proof hardware devices to prevent it from being taken out of the devices for impersonation and other malicious activities.

In IoT solution deployments, it is important to check the identity of the device that is communicating with the messaging gateway. For the first time running demo, TPM will generate key pairs for the devices, which are then used to authenticate and encrypt the traffic. The keys are generated inside the TPM itself and are thereby protected from being retrieved by external programs. In fact, even without harnessing the capabilities of a hardware root of trust and secure boot, the TPM is also valuable just as a hardware key store. The private keys are protected by the hardware and offer far better protection than a software key. This integration uses the PKCS#11 protocol as the interface to the TPM.

After generating key pairs, this demo uses the FreeRTOS MQTT library to connect to the AWS Cloud and then periodically publish messages to an MQTT topic hosted by the AWS IoT MQTT broker. A specific Android application developed by SHC also uses this topic to communicate with Corvette-F1 board to control its on-board LEDs.

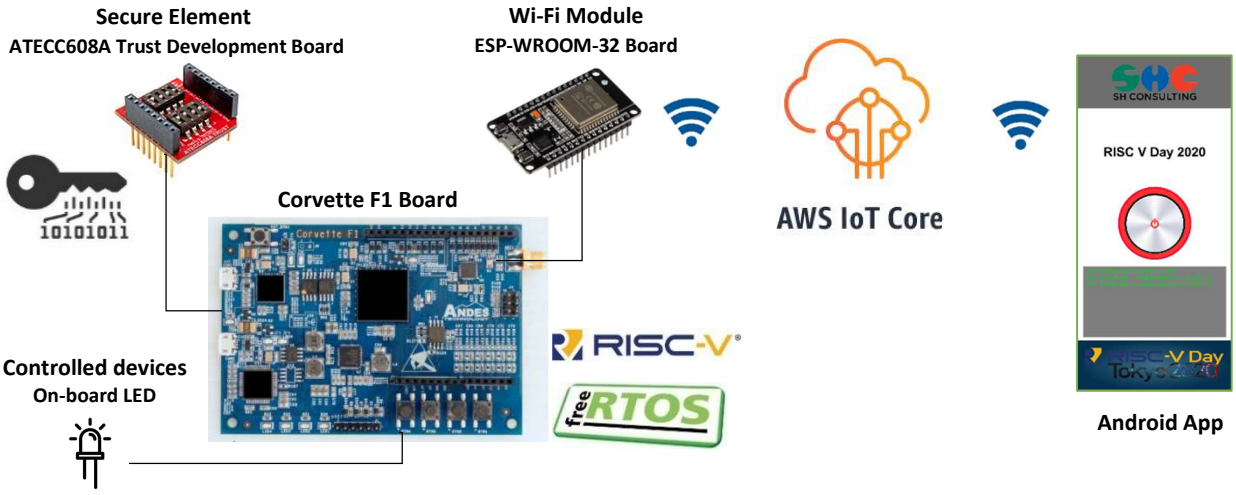


Figure 1: Corvette F1 Root of Trust and AWS IoT demonstration

Corvette F1 Board

General Description

Andes Technology is a leading supplier of high-performance low-power compact 32/64-bit RISC-V CPU cores and a founding member of the RISC-V Foundation. Its Corvette-F1 N25 platform is one of the first RISC-V platforms qualified for Amazon FreeRTOS. Amazon FreeRTOS is an open source operating system for microcontrollers from Amazon Web Services (AWS) that makes small, low-power edge devices easy to program, deploy, secure, connect, and manage. Developers can take advantage of Amazon FreeRTOS features and benefits by using the RISC-V platform from Andes Technology.

As more and more technologies have been deployed to the internet, the IoT market grows with a wide variety of diversified applications. The RISC-V Instruction Set Architecture (ISA) provides enhanced flexibility, extensibility, and scalability that can help generate new possibilities for the IoT and making it easier to design compact IoT hardware to take advantage of this growing market. By combining the RISC-V platform with solutions like Amazon FreeRTOS, AWS IoT Greengrass, and AWS IoT Core, Andes Technology can help developers to create comprehensive and competitive RISC-V-based IoT systems.

The Corvette-F1 N25 platform is one of the first RISC-V platforms qualified for Amazon FreeRTOS. The Corvette-F1 N25 platform is an FPGA-based Arduino-compatible evaluation platform. It comes with a 32-bit RISC-V AndesCore™ N25 running at 60MHz, 4MB Flash, 256KB instruction SRAM and 128KB data SRAM, and AndeShape™ AE250 Platform IP with a rich set of peripherals including GPIO, I2C, PWM, SPI, and UART. Users can easily build the prototypes and applications of IoT devices under Arduino standard IDE and full-featured AndeSight™ IDE.

ESP WROOM-32 Wifi Module

ESP32-WROOM-32 is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks

ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I²S and I²C.

In this demo we use UART to send AT command to ESP WROOM 32 to enable wifi connection

ATECC608A Chip

Features

• Cryptographic Co-Processor with Secure Hardware-based Key Storage:
  – Protected Storage for up to 16 Keys, Certificates or Data
• Hardware Support for Asymmetric Sign, Verify, Key Agreement:
  – ECDSA: FIPS186-3 Elliptic Curve Digital Signature
  – ECDH: FIPS SP800-56A Elliptic Curve Diffie-Hellman
  – NIST Standard P256 Elliptic Curve Support
• Hardware Support for Symmetric Algorithms:
  – SHA-256 & HMAC Hash including off-chip context save/restore
  – AES-128: Encrypt/Decrypt, Galois Field Multiply for GCM
• Networking Key Management Support:
  – Turnkey PRF/HKDF calculation for TLS 1.2 & 1.3
  – Ephemeral key generation and key agreement in SRAM
  – Small message encryption with keys entirely protected
• Secure Boot Support:
  – Full ECDSA code signature validation, optional stored digest/signature
  – Optional communication key disablement prior to secure boot
  – Encryption/Authentication for messages to prevent on-board attacks
• Internal High-Quality NIST SP 800-90A/B/C Random Number Generator (RNG)
• Two High-Endurance Monotonic Counters
• Guaranteed Unique 72-bit Serial Number
• Two Interface Options Available:
  – High-speed Single Pin Interface with One GPIO Pin
  – 1 MHz Standard I2C Interface
• 1.8V to 5.5V IO Levels, 2.0V to 5.5V Supply Voltage
• <150 nA Sleep Current
• 8-pad UDFN and 8-lead SOIC Packages

Applications

• Network/IoT Node Endpoint Security

Manage node identity authentication and session key creation & management. Supports the entire ephemeral session key generation flow for multiple protocols including TLS 1.2 (and earlier) and TLS 1.3

• Secure Boot

Support the MCU host by validating code digests and optionally enabling communication keys on success. Various configurations to offer enhanced performance are available.

• Small Message Encryption

Hardware AES engine to encrypt and/or decrypt small messages or data such as PII information. Supports AES-ECB mode directly. Other modes can be implemented with the help of the host microcontroller. Additional GFM calculation function to support AES-GCM.

• Key Generation for Software Download

Supports local protected key generation for downloaded images. Both broadcast of one image to many systems, each with the same decryption key, or point-to-point download of unique images per system is supported.

• Ecosystem control and Anti-Counterfeiting

Validates that a system or component is authentic and came from the OEM shown on the nameplate.

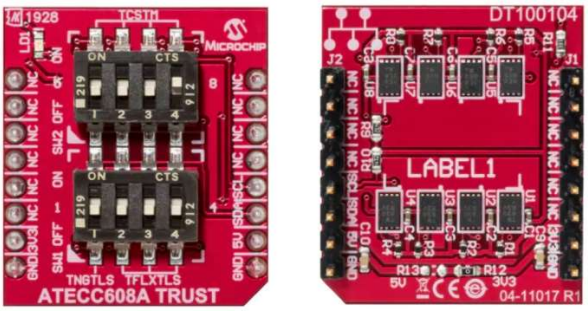


Figure 2: ATECC608A Trust Development Board (top view and bottom view)

Cryptographic Operation

The ATECC608A implements a complete asymmetric (public/private) key cryptographic signature solution based upon Elliptic Curve Cryptography and the ECDSA signature protocol. The device features hardware acceleration for the NIST standard P256 prime curve and supports the complete key life cycle from high quality private key generation, to ECDSA signature generation, ECDH key agreement, and ECDSA public key signature verification.

The device is designed to securely store multiple private keys along with their associated public keys and certificates.

Random private key generation is supported internally within the device to ensure that the private key can never be known outside of the device. The public key corresponding to a stored private key is always returned when the key is generated and it may optionally be computed at a later time.

The ATECC608A can generate high-quality random numbers using its internal random number generator, that is designed to meet the requirements documented in the NIST 800-90A, 800-90B and 800-90C documents.

These random numbers can be employed for any purpose, including usage as part of the device's crypto protocols. Because each random number is guaranteed to be essentially unique from all numbers ever generated on this or any other device, their inclusion in the protocol calculation ensures that replay attacks (i.e. re-transmitting a previously successful transaction) will always fail.

Root of Trust with Microchip's Trust Platform

In the age of the IoT, hardware-based security is the only way to protect secret keys from physical attacks and remote extraction. However, extensive security expertise, development time, and costs are required to configure and provision each device. Microchip has developed a key security service, called Microchip's Trust Platform for its CryptoAuthentication family, that provides an easy way for OEMs, small or large, to implement secure authentication for hardware devices. The service provides a secure system to manufacture device keys and a supply chain to generate custom part numbers

Based on the ATECC608A secure element, the Trust Platform is available in three tiers to meet the needs of users ranging from out-of-the-box pre-provisioned to fully customizable. The ATECC608A provides Common Criteria Joint Interpretation Library (JIL) "high"-rated secure key storage, giving customers confidence that devices implement industry-proven security practices and the highest level of secure key storage. Hardware-based root of trust storage and cryptographic countermeasures protect the device against the widest classes of known physical attacks.

The three tiers are Trust&GO, TrustFLEX, and TrustCUSTOM.

- Trust&GO (ATECC608A-TNGTLS) for the mass market, provides zero-touch pre-provisioned secure elements with a minimum orderable quantity (MOQ) as low as 10 units. Device credentials are pre-programmed, shipped and locked inside the ATECC608A for automated cloud or LoRaWAN authentication onboarding.
- TrustFLEX (ATECC608A-TFLXTLS) offers the flexibility to use the customer's certificate authority while benefiting from pre-configured use cases. These use cases include baseline security measures such as Transport Layer Security (TLS) hardened authentication for connecting to any IP-based network using any certificate chain, LoRaWAN authentication, secure boot, Over-the-Air (OTA) updates, IP protection, user data protection, and key rotation.
- TrustCUSTOM (ATECC608A-MAHDA) enables complete customization, providing customer-specific configuration capabilities and custom credential provisioning.

Microchip worked with Amazon Web Services (AWS) to enable a straightforward and simplified on-board process into AWS IoT services for products designed with all variants of the Microchip Trust Platform. The ATECC608A secure element can be paired with any microcontroller and microprocessor.

Amazon IoT Core

Easily and securely connect devices to the cloud. Reliably scale to billions of devices and trillions of messages.

What is AWS IoT Core?

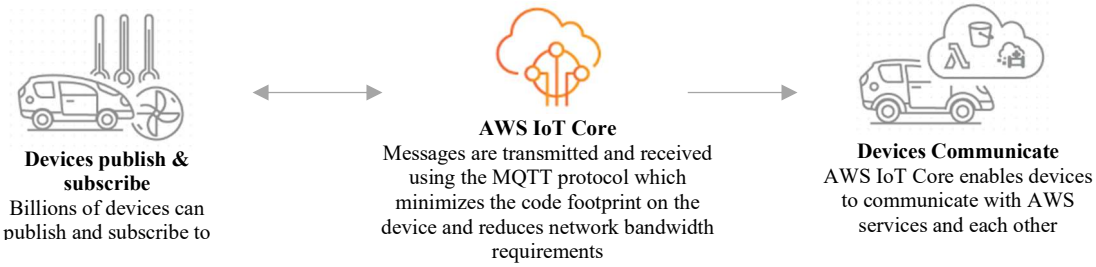
AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT Core can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely. With AWS IoT Core, your applications can keep track of and communicate with all your devices, all the time, even when they aren't connected.

AWS IoT Core also makes it easy to use AWS and Amazon services like AWS Lambda, Amazon Kinesis, Amazon S3, Amazon SageMaker, Amazon DynamoDB, Amazon CloudWatch, AWS CloudTrail, Amazon QuickSight, and Alexa Voice Service to build IoT applications that gather, process, analyze and act on data generated by connected devices, without having to manage any infrastructure.

How does AWS IoT Core work?

Connect and manage your devices

AWS IoT Core allows you to easily connect any number of devices to the cloud and to other devices.



Devices publish & subscribe
Billions of devices can publish and subscribe to

AWS IoT Core
Messages are transmitted and received using the MQTT protocol which minimizes the code footprint on the device and reduces network bandwidth requirements

Devices Communicate
AWS IoT Core enables devices to communicate with AWS services and each other

Secure device connections and data

AWS IoT Core provides automated configuration and authentication upon a device's first connection to AWS IoT Core, as well as end-to-end encryption throughout all points of connection, so that data is never exchanged between devices and AWS IoT Core without proven identity. In addition, you can secure access to your devices and applications by applying policies with granular permissions.

Input
An array of temperature sensors transmit data

Authenticate
The connection to AWS IoT Core is authenticated

AWS IoT Core
If the sensors agree the temperature is above a threshold, they turn on the fan. Only authenticated Users can control the fan

Authenticate
The connection to Fan is authenticated

Output
The fan receives a Command and turns on

Read and set device state at any time

AWS IoT Core stores the latest state of a connected device so that it can be read or set at anytime, making the device appear to your applications as if it were online all the time.

Input
Connected mass spectrometer reports its state and reading though out multi-hour cycle

Intermittent Connection

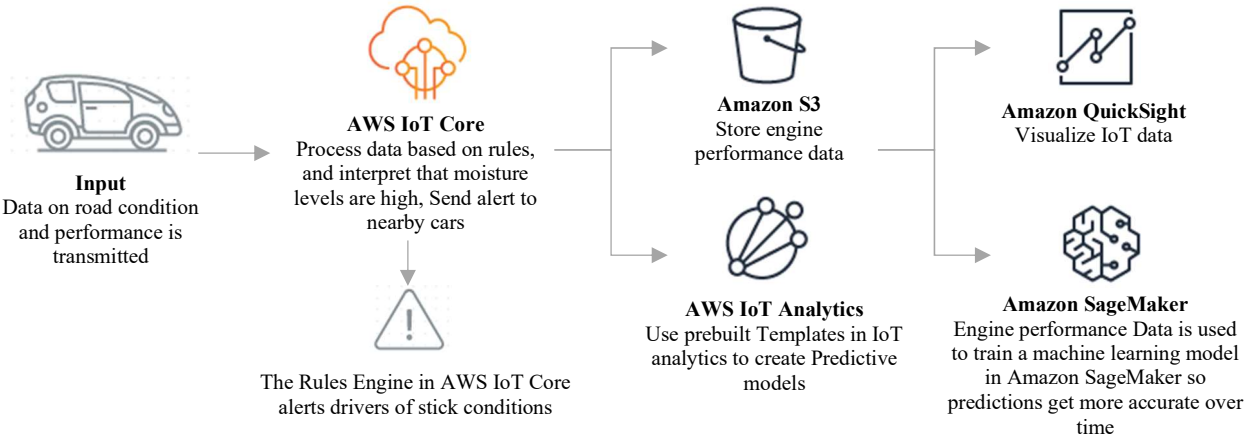
AWS IoT Core
The spectrometer goes offline when its cycle completes, but its last-reported state persists in AWS IoT Core

REST APIs

Output
Technicians can use mobile apps to set new desired states (e.g. pause the cycle), or query the last reported state of the spectrometer

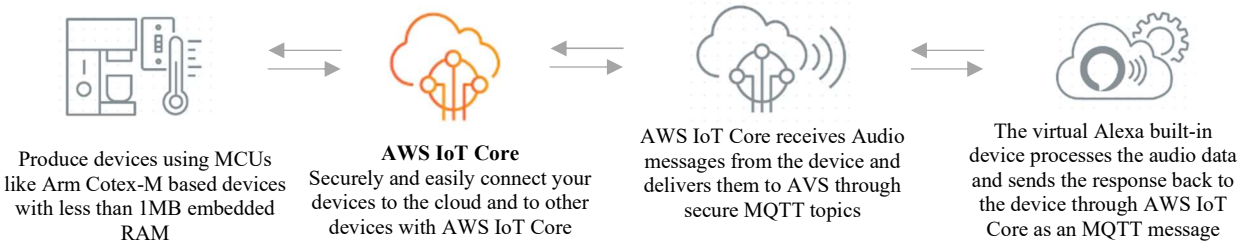
Process and act upon device data

With AWS IoT Core, you can filter, transform, and act upon device data on the fly, based on business rules you define.



Cost-effectively scale to hundreds of millions of Alexa Built-in devices

The Alexa Voice Service (AVS) Integration for AWS IoT Core introduces a new virtual Alexa Built-in device in the cloud. It allows customers to send and receive audio messages over the reserved MQTT topics, interface with the device microphone and speaker, and manage the device-side state all while using the same secure IoT Core connection.



Setup and show demo

Basically we need to following below steps:

1. Create the Signing CA
2. Create the Device Certificate
3. Register with AWS IoT
4. Run the demos

Below is descripted more detail:

Step 1: Create the Signing CA

1. Create the Root CA

First, create your Root CA.
The Root CA would be the trust anchor from which the whole chain of trust is derived.
Microchip has created a tool to create a Root CA using Python.
We will use this tool to create our Root CA.

2. Create the Signing CA

The Signing CA is an intermediate CA and is issued by the Root CA.
It will be used to issue your device certificate.
The purpose of creating and using Intermediate CA is primarily for security.
Because if the intermediate private key is compromised,
then the Root CA can revoke the intermediate certificate and create a new one with a new cryptographic key pair.

This CA certificate will be registered with your AWS IoT account.
In order to prove your ownership of the Signing CA, obtain a registration code from AWS.
This registration code will be used to generate a verification certificate (by placing the code into the issued certificate as its subject name).

3. Run “ca_create_signer.py” script to generate the Signing CA certificate & keys.

Microchip also create a tool to generate a Signing CA.
Run this script, we could generate the Signing CA certificate & keys, as well as the verification certificate.
The Signing CA certificate & keys will be used to issue your device certificate.
The Signing CA certificate is an X.509 certificate that contains the public key and is signed by the Root CA we just created.
The verification certificate is used to register your signing CA certificate with AWS IoT in the next step.
d. Register your CA certificate with AWS IoT

Because we are using a self-signed Root CA, so AWS IoT doesn't recognize it.
Therefore, we need to register the CA certificate with AWS IoT.

Now register your CA certificate with AWS using AWS CLI.
AWS CLI will send the signing CA certificate to AWS to register that cerfiticate with AWS.
It also send the verification certificate we created above to prove that you own the private key associated with the CA certificate.
AWS CLI is using HTTPS to send data to AWS, so the data has been encrypted.
The private key won't be sent to AWS.

4. This API returns the certificate ID for your new CA certificate.

Certificate ID is the 64 character hex value.
It is used to identify the certificate from all the certificates registered on AWS IoT.
The ID won't be added to the device's X509 certificate.

f. Your signing CA certificate is now “inactive”, meaning that it cannot yet be used. “Activate” now!

By default the certificate is in "INACTIVE" state when we register it with AWS.
It means that the certificate is in AWS server already, but it cannot be used yet.
We need to activate it.

5. You have now created your Signing CA, and registered it with your AWS account.

Step 2: Create the Device Certificate

1. Go to the FreeRTOS code, and open the demo project.

“Demos¥dev_mode_key_provisioning¥src

Aws_dev-mode_key_provisioning.c”file

aws_dev-mode_key_provisioning.c is used to provision key and certificate for FreeRTOS.
Without ATECC608A, the key pair is generated using software and is saved in flash memory.
So it could be leaked.
When using ATECC608A, the key pair is generated in ATECC608A.
The public key and device serial number can be exported from ATECC608A.
But the private key will never leaves the ATECC608A.

ATECC608A uses Elliptic-curve cryptography (ECC) as shown in its name.
ECC has some advantages:
* Shorter keys are as strong as long key for RSA.
* Low on CPU consumption.
* Low on memory usage.

2. Change macro “keyprovisioingFORCE_GENERATE_NEW_KEY_PAIR” to 1

Changing this macro to 1 to force the device to generate new key pair each time it runs.
The public key and device serial number can be exported from ATECC608A.
We need these information to create Device Certificate.

3. Build the FreeRTOS project.

4. Burn the firmware to MCU

Flash the firmware aws_demos.bin we just built to MCU.

5. Reset

Reset the MCU to boot up the new firmware.
The FreeRTOS will be booted up here.

When the device boots it will generate a private key (which never leaves the ATECC608A), as well as a public key.

6. The public key and device serial number are exported from the ATECC608A and printed to the console.

After the key pair is generated in ATECC608A, MCU will get the public key and device serial number from ATECC608A and print them out.
The serial number is the 18 characters following CN=.
The public key is 91 bytes in hex format printed in 6 lines.

7. Copy the six lines of public key bytes into a file called
*DevicePublicKeyAsciiHex.txt

This file contains only the public key.

8. Currently you can see the system can't connect to AWS

Currently, the device is using the newly created key pair to communicate with AWS.
But AWS does not have any information about the new key pair yet.
So the device could not establish the connection to AWS.

Now we will create the Device Certificate with the public key and device serial number and sign it with the signing CA we created before.
The Python tool to create device cerfiticate need the public key in PEM format.
So we need to convert the public key from hex numbers to PEM file.
The easiest way to accomplish this is convert the public key from hex numbers to DER format using xxd tool.
Then convert the public key from DER format to PEM format using openssl tool.

What are the DER and PEM format?
DER (Distinguished Encoding Rules): Binary encoding for certificate data.
PEM (Privacy-enhanced Electronic Mail): The base64 encoding of the DER-encoded certificate, with a header and footer lines added.
They are two main methods for encoding certificate data.

9. Use the command-line tool xxd to convert the public key to DER format

This step will generate the DevicePublicKeyDer.bin file.
It is the public key in DER format.

10. Use openssl to convert public key from DER format to PEM format

This step will generate the public_key.pem file.
It is the public key in PEM format.

11. Change macro "keyprovisioingFORCE_GENERATE_NEW KEY_PAIR" to 0

Don't forget to disable the temporary key generation setting you enabled above.
Otherwise, the device will create another key pair, and you will have to repeat the previous steps.

12. Create a device certificate using the ca_create_device script

Use the 18 characters serial number for the thing name, and the public_key.pem file that you formatted as your public key.

You now have a device certificate called device.crt.
This cerfiticate will be used to authenticate our device with AWS.

Step 3: Register with AWS IoT

Now when the device cerfiticate has been created, let's setup our device with AWS IoT.

1. Register your device certificate with AWS IoT

This step lets AWS know that your device certificate is linked to your account,
and that the Signer CA you previously created is above your device certificate in the chain of trust.
The output is the certificate ID of your device certificate.
Take note of your certificate ID, as it will be used later in the instructions when we attach a policy to it.

2. Activate your certificate

The certificate is in "INACTIVE" state by default when we register it.
Now, use AWS CLI to activate it.

3. Create a policy called “DemoPolicy” for your thing’s certificate

An AWS IoT policy is a JSON document that indicates the access right to AWS IoT service.
AWS IoT policy is used to authorize your device to perform AWS IoT operations, such as subscribing or publishing to MQTT topics.
Your device presents its certificate when sending messages to AWS IoT.
To allow your device to perform AWS IoT operations, you must create an AWS IoT policy and attach it to your device certificate.

For this demo, we will create a policy to allow our device to Connect/Publish/Subscribe/Receive to AWS IoT.
In this video, the policy has been created before, so we don't need to recreate it.

4. Attach the policy to your device’s certificate

When we attach a policy to a certificate, we has set the access right for the certificate.
It means that the device which uses this certificate will have the access right that defined in the policy.

5.Create a Thing - this is the digital representation of your device in AWS IoT

An IoT thing is a representation and record of your physical device in the AWS IoT cloud.
Any physical device needs a thing record in order to work with AWS IoT.

We will create a thing with the thing name is 18 characters serial number obtained from ATECC608A.

6. Attach your device certificate to your thing

Now associate the device certificate with the thing we have just created.

The certificate is now ready to be used.

7. Export the client_credential_keys.h file

Next you will need to tell your device about the certificate you just created for it.
Generate a header file to include your device certificate and signer CA certificate in your demo project.

This step creates the aws_clientcredential_keys.h file that is used by FreeRTOS to save the generated certificates into ATECC608A.

8. Get the endpoint of AWS IoT

To connect programmatically to an AWS service, you use an endpoint.
An endpoint is the URL of the entry point for an AWS web service.
The device need to connect to AWS IoT, so we need to get the endpoint of AWS IoT.

9. Update aws_clietncredential.h file

Now enter the AWS IoT endpoint and the Thing Name into aws_clietncredential.h.
This file is used by FreeRTOS to indicate:
* Which endpoint it will connect to.
* Which Thing it will associate with.

10. Build again

Now build the FreeRTOS project again with the new aws_clietncredential.h and aws_clientcredential_keys.h.

11. Burn to MCU

Flash the new firmware we have just built to the MCU.
And reset the MCU to boot up the new firmware.

Step 4: Run The demos

1. Now we can connect to AWS IoT server and publish and subscribe to control LEDs

The device has successfully connected to AWS IoT server using the key pair from ATECC608A now.
In this demo, it will subscribe to a MQTT topic.
It will control the LEDs according to the command it receives via this MQTT topic.

We will use an Android phone to send command to this MQTT topic.
Let's try it.

Press button on Android phone to turn the LEDs on.
The LEDs are on now.

Press button on Android phone again to turn the LEDs off.
The LEDs are off now.

About SH Consulting Group

SH Consulting Group (SHC) has engineers in US, Vietnam and in Japan specialized in providing stability to RTOS, device drivers, and wireless connectivities for MCUs such as H8s, SHs, ARMs and RISC-Vs. It has been integrating OSes such as QNX, .NETMF, Linux, and Windows for MCUs and wireless solutions such as Lora, WiFi and Bluetooth for many years. They worked on Windows, Android and iOS platforms. In recent years SHC engineers enabled FreeRTOS for large semiconductor companies on ARM platforms and direct this effort to RISC-Vs.

SH CONSULTING K.K. (JAPAN)
Tokyo Head Office: 7-18-13-502 ginza, Chuo-ku, Tokyo, Japan 104-0061
Phone: 03-3833-3717
Tokyo Design Center: Yamamoto Building 5F, 3-23-12 Minami-cho, Kokubunji-shi, Tokyo 185-0021, Japan

SH CONSULTING VIETNAM COMPANY LTD. (VIETNAM)
(Local Name: CÔNG TY TNHH SH CONSULTING VIỆT NAM)
Quang Trung Software park, Tan Chanh Hiep
Ward, District 12 Ho Chi Minh City
Phone: 84-8-3715-0060

SOFTWARE HARDWARE & CONSULTING LLC (USA)
San Francisco Bay Area USA:
Software Hardware & Consulting LLC
1325A Church St, San Francisco, CA 94114-3900
Phone: +1-(408) 510-8221