

EmSPARK Suite

AWS IOT Core for HelmsDeep96 Board

Date November 15, 2019 | Version 1.0



CONFIDENTIAL AND PROPRIETARY

THIS DOCUMENT IS PROVIDED BY SEQUITUR LABS INC. THIS DOCUMENT, ITS CONTENTS, AND THE SECURITY SYSTEM DESCRIBED SHALL REMAIN THE EXCLUSIVE PROPERTY OF SEQUITUR LABS, ARE CONFIDENTIAL AND PROPRIETARY TO SEQUITUR LABS, AND SHALL NOT BE DISCLOSED TO OTHERS.

TABLE OF CONTENTS

AWS IOT Core for HelmsDeep96 Board	1
1. Introduction.....	3
1.1. Prerequisites	3
1.2. EmSPARK Suite Package Contents	3
2. Installation Procedure	3
2.1. Installing the filesystem on SDCARD	4
2.2. Flashing the Secure Bootloader.....	4
2.3. Starting the Console	5
2.4. Secure Boot Mode - Starting and Using the System	5
2.5. Re-flashing a Board Already in Secure Mode.....	6
3. CoreLockr Libraries.....	7
3.1. Suite Contents.....	7
3.2. Preloaded Keys and Certificates in the TEE	7
4. TLS Mutual Authentication and Session Establishment with Amazon AWS IoT	8
4.1.1. Linux Environment: Generate the AWS Custom CA and AWS Device Certificates	10
4.1.2. AWS Console and Board: Setup AWS IoT for the Example	10
4.1.3. AWS Console: Create a Publish/Subscribe Policy	15
4.1.4. AWS Console: Define the Topic to Subscribe.....	16
4.1.5. Linux Environment: Configure the AWS Example Application that Connects to AWS	17
4.1.6. Board: Save the Certificates and Key in the TEE	18
4.1.7. Board: Execute the AWS Example Application	19
4.1.8. Board and AWS Console: Observe Published Messages.....	19
Appendix A: Secure SAM-BA Loader Instructions for Windows	22
Appendix B: Secure SAM-BA Loader Instructions for Linux.....	27

1. INTRODUCTION

The **EmSPARK Suite - Evaluation Kit, Getting Started** guide provides an overview of the prerequisites to use the Evaluation Kit and the Kit package contents. It also provides information to flash the secure bootloader on the board, install the file system on SDCARD and start the system. After completing this guide, see the [USER_GUIDE.pdf](#) tutorial for a description of the Evaluation Kit and the [CORELOCKR_LIBRARIES_GUIDE.pdf](#) for instructions to build and run the provided example applications.

1.1. Prerequisites

The guide assumes that the following hardware and software are available:

- HelmsDeep96 board
- Linux system to extract the Evaluation Kit package and build the example applications
- Windows system or Linux system to flash the Secure Bootloader
- SDCARD to install the filesystem
- Two micro USB cables to connect the board a computer

1.2. EmSPARK Suite Package Contents

Download the Evaluation Kit package. Expand the package in a Linux environment:

```
tar -zxvf security_suite_eval_HelmsDeep_[release].tar.gz
```

Extracting the tar file creates the following file structure:

- `corelockr`, the CoreLockr libraries, example applications and API documentation
- `coretee_dev_kit`, the toolchain and the client API needed for building the example applications
- `eval_kit_loader`, the Secure SAM-BA Loader and firmware
- `filesystem`, the filesystem to be installed on the SDCARD
- `GETTING_STARTED.pdf`, guide instructing the Evaluation Kit installation process
- `USER_GUIDE.pdf`, an overview of the Evaluation Kit
- `CORELOCKR_LIBRARIES_GUIDE.pdf`, an overview of the CoreLockr libraries and tutorial to build and execute the example applications
- `COPYRIGHT.txt`, the copyright notice
- `RELEASE_NOTES.txt`, information about the release

2. INSTALLATION PROCEDURE

The process consists of these steps, which will be detailed in the following sections:

- Install the filesystem on an SDCARD
- Flash the secure bootloader, CoreTEE and Linux Kernel on the HelmsDeep96 board
- Start the console
- Start the system

2.1. Installing the filesystem on SDCARD

To install the board's Normal World filesystem on an SDCARD, partition and create an ext4 filesystem on the first partition on the SDCARD. Then extract the filesystem in `filesystem/rootfs_seq_[release].tar.gz` directly into the root of the card. The SDCARD must be 2GB or larger.

The following steps are one way to install the filesystem on the SDCARD (tested on 16.04.2):

- Identify the card device in the system and unmount it.
- Create a partition, which can use the entire SDCARD:

```
fdisk /dev/<device>
```

- Create an ext4 filesystem on the SDCARD first partition:

```
mkfs -t ext4 /dev/sd[x]1
```

where `/dev/sd[x]1` is the card device in the system, partition 1.

- Mount the empty card, usually in a mount-point:

```
mount /dev/sd[x]1 /mnt/sdcard/
```

where `/mnt/sdcard/` is a mount-point previously created in the system.

- Change to the mount-point and extract there `rootfs_seq_[release].tar.gz`. To preserve permissions, execute this step as root:

```
cd /mnt/sdcard
```

```
tar -zxvf /security_suite_eval_
HelmsDeep_[release]/filesystem/rootfs_seq_[release].tar.gz
```

- After the tar file is extracted, execute "`sync`". This step can take from a few seconds to a few minutes.
- Unmount and eject the SDCARD from the Linux system.
- The SDCARD is now ready to be inserted on the board.

2.2. Flashing the Secure Bootloader

Note: After flashing the board with the EmSPARK Suite Evaluation Kit, because the Evaluation Kit uses emulated fuses stored on the chip SRAM, in order to maintain their values, it is recommended to keep the board powered up. If the board is unpowered, the provisioning data will be eased and as a result the board will need to be provisioned again. This will not affect the SDCARD data.

If the memory is erased, the behavior during boot will return to the default boot configuration.

The Secure Bootloader can be flashed on the board in either Windows or Linux environments. The `eval_kit_loader` directory contains everything necessary to flash the bootloader, including the necessary scripts for Windows and Linux environments, the loader, and documentation:

- `sam-ba_3.3_linux_x64` and `sam-ba_3.3_win32` are the Secure SAM-BA Loader tools for Linux and Windows environments.
- `flash_eval_kit.sh`, in Linux environment, flashes the firmware to the device.
- `flash_eval_kit.bat`, in Windows environment, flashes the firmware to the device.

To flash the bootloader:

- The flashing procedure does not require pre-installation of software.
- Follow the board flashing steps for your work environment:
 - Windows environment: see

- **Appendix A: Secure SAM-BA Loader Instructions for Windows** for step-by-step guide.
- Linux environment: see **Appendix B: Secure SAM-BA Loader Instructions for Linux** for instructions.

NOTE: To flash the board, use a native Windows or Linux machine, not a virtual machine.

Flashing the board on a virtual machine may fail.

2.3. Starting the Console

To start the serial console which is the TEE console where occasionally the Secure World prints output messages:

- Connect a micro USB cable to J10 (debug) micro USB port on the board
- Connect the host machine to the serial port
 - 115200 bps
 - No parity
 - 8 bits
 - 1 stop bit
 - No flow control
- Connect a micro USB cable to the PC/power micro USB port on the board

2.4. Secure Boot Mode - Starting and Using the System

To start and use the system on the board:

- Insert the SDCARD on the board (the SDCARD may be inserted while the power is off)
- Start the console

After the board starts up:

- The console shows "`Default username:password = root:root`" and the user is prompted to enter access credentials.
- The board is configured to acquire an IP address using DHCP

The board is ready for your configuration:

- Required configuration:
 - The date on the board must be current for certificate management. When the board is used offline, the date must be configured. When the board is configured for remote access, verify that the date is current.
- Optional configuration:
 - Configure user(s). Only users in the `coretee` group have access to the TEE clients, therefore only users in the `coretee` group can execute applications using the CoreLockr APIs. If users are created to execute the example applications, they must be added to the `coretee` group.

- Configure the board for remote access. The board is configured to acquire an IP address using DHCP. SSH is set up in the filesystem.

The board set up is complete. You can transfer to the board and execute example applications that use the EmSPARK Suite APIs.

2.5. Re-flashing a Board Already in Secure Mode

To re-flash a board that had been previously flashed and it is already in secure mode:

- Remove power from the board
- Wait for at least 30 seconds
- Reapply power the board

This clears any memory and the secure state of the board. The board should now be ready to be re-flashed using the steps in Section **2.2 Flashing the Secure Bootloader**.

3. CORELOCKR LIBRARIES

This document presents an overview of CoreLockr APIs' capabilities and explains the example for setting up the AWS IOT Core included in the EmSPARK Suite. For information about the EmSPARK Suite, please see the **User Guide**.

3.1. Suite Contents

- APIs and Normal World Assets
 - CoreLockr APIs (C libraries)
 - OpenSSL Crypto Engine
 - Code Examples
 - Toolchain and Client API
- CoreTEE, Secure OS required to access TrustZone secured resources
- Trusted Applications
 - Secure Storage TA
 - Secure Certificates TA
 - Crypto TA

On the board, only users who are members of the `coretee` group can execute applications using the CoreLockr APIs.

3.2. Preloaded Keys and Certificates in the TEE

After the device provisioning, to support the execution of example applications or the development of other test applications, the following keys and certificates are loaded in the TEE. With exception of the Device Key and Device Certificate which are unique per device, all keys are the same in all kits and are not meant to protect any secrets but to be used with the examples. Table 1 lists the preloaded certs and keys.

Table 1 - Preloaded Keys and Certificates

Cert/Key	Name of cert/key exposed by Security suite	Description
Cloud IoT Root CA	CLRSC_CLOUD_CERT	Root CA of the Cloud IoT. It is used for TLS mutual authentication.
Cloud IoT Public Key	CLRC_CLOUD_PUBLIC_KEY	Public key extracted from Cloud IoT Root CA.
OEM Root Certificate	CLRSC_OEM_CERT	OEM Root Certificate. For the Evaluation Kit this cert is predefined. The associated private key is provided as a file for execution of the payload verification example application

OEM Public Key	CLRC_OEM_PUBLIC_KEY	Public Key extracted from OEM certificate. It is used for authentication and integrity checking (payload verification). It is also used to authenticate TEE Certificate Management commands.
Device Private Key	CLRC_DEVICE_PRIVATE_KEY	Device private key created in device TEE during provisioning. It is used for TLS connection establishment.
Device Certificate	CLRSC_DEVICE_CERT	Unique device certificate created at flashing signed with OEM Private Key. It is used for TLS mutual authentication.
Device Public Key	CLRC_DEVICE_PUBLIC_KEY	The pair of the Device Private key.

4. TLS MUTUAL AUTHENTICATION AND SESSION ESTABLISHMENT WITH AMAZON AWS IoT

Amazon cloud allows customers to use device certificates signed and issued by their own certificate authority (CA) to connect and authenticate with AWS IoT. This is an alternative to using certificates generated by AWS IoT and better fits customers' needs. This method is used by "things" using MQTT protocol. MQTT is using TLS as a secure transport mechanism. In IoT each "thing" needs to be uniquely identified by the cloud application and that is realized by using device certificates as identifiers.

During TLS connectivity establishment, AWS IoT authenticates the connecting device by extracting the device certificate and verifying its signature against a customer preloaded root certificate. Similarly, the device needs to verify the server certificate against the stored AWS IoT root certificate to confirm the authenticity of the server to which it connects. TLS mutual authentication requires the device to prove the ownership of its private key used to form the device certificate and this is being done by signing some data packets with the private key.

The EmSPARK Suite facilitates the creation and signing of a device certificate during the firmware flashing. The device certificate is intended for establishing TLS connections with mutual authentication. The Evaluation Kit provides an example of how to use the device certificate and TrustZone based crypto for establishing a TLS connection with Amazon AWS IoT (usually used for running MQTT protocol that runs on top of TLS).

The example requires the user to create an AWS account, create an OEM Root CA and upload it to AWS. The Device Certificate needs to be signed with the private key that created the OEM Root CA so the two certificates are chained. Due to the fact that Amazon AWS does not allow the activation of the same OEM Root CA for multiple AWS accounts and the Evaluation Kit is the same for all users, the example provided is not using the certificate created at firmware flashing.

Instead, the example guides the user to perform the following steps: create an AWS Custom CA, register the Custom CA in AWS, provide verification to AWS and create an AWS Device Certificate. Then save the created AWS Custom CA and AWS Device key and cert in TrustZone. This way the AWS CAs and the AWS Device Certificate are unique and can be used with AWS for the Evaluation Kit by multiple users.

To set and test the TLS mutual authentication and connectivity to AWS IoT, for this example, users generate their own AWS Custom CA private key and certificate and AWS device key and certificate, which must be ECDSA 256. The Evaluation Kit provides the `aws_openssl_generate_certs.sh` script that may be used to generate them, and the `aws_management` tool to save them in the TEE. The AWS cloud certificate is preloaded in the TEE. The example uses the following keys and certificates:

1. **AWS IoT Root Certificate**, `CLRSC_CLOUD_CERT` preloaded in the TEE.
2. **AWS Custom CA Key**, this private key is generated by the user. It is used to sign the AWS Device Certificate and to complete the AWS Custom CA Certificate registration process with AWS IoT.

Note: the AWS Custom CA key with the Evaluation Kit is used only for fulfilling testing; the Production Kit does not require the use of the AWS Custom CA key and the device certificate is created and signed during flashing procedure.

3. **AWS Custom CA Certificate**, the user creates this certificate and using the provided tool saves it in the TEE. This certificate needs to be uploaded to the AWS IoT cloud.
4. **AWS Device Key**: private key generated by the user. Using the provided tool, the user saves it in the TEE.
5. **AWS Device Certificate**, the user creates this certificate. It must be created and signed with the AWS Custom CA Key. It is used during the AWS device registration step. Using the provided tool, the user saves it in the TEE.

Note: For the Evaluation Kit only, the AWS Device Certificate is created by the user; for Production Kit the Device Certificate is created, signed and saved during the flashing procedure.

Steps for the example execution described in the following sections:

- Linux environment: Generate the required keys and certificate
- AWS IoT: Setup the AWS IoT for the example
- AWS console: Create a publish/subscribe policy
- AWS IoT: Identify an AWS topic to route messages
- Linux environment: Configure the AWS example application that connects to AWS
- Linux environment: Build the application AWS example application
- On the board: Save in the TEE the generated keys and certificate

- On the board: Execute the AWS example application
- AWS IoT console: Observe published messages

4.1.1. Linux Environment: Generate the AWS Custom CA and AWS Device Certificates

The Evaluation Kit provides a script that can be used to generate these certificates, `aws_openssl_generate_certs.sh`. The script uses configuration files for customization. The script and configuration files are in `corelockr/examples/AWS/tools/aws_generate_certs.tar.gz`. Expand the tar file and follow the steps described below.

Configure the AWS Custom CA Certificate

In `corelockr/examples/AWS/tools/conf/ca.conf` configure the AWS Custom CA certificate. Note that the example application requires that all keys and certs be ECDSA 256. Those parameters should not be changed to avoid errors during the application execution.

Configure the AWS Device Certificate

In `corelockr/examples/AWS/tools/conf/derived.conf` configure the AWS Device Certificate information. The same requirement about ECDSA 256 keys applies.

Generate Keys and Certificates

Executing `./aws_openssl_generate_certs.sh` generates the following keys and certificates:

- **AWS Custom CA Key:** `aws_custom_ca_key.pem`
- **AWS Custom CA Certificate:** `aws_custom_ca_cert.pem`
- **AWS Device Key:** `aws_device_key.pem`
- **AWS Device Certificate:** `aws_device_cert.pem`
- `aws_device_cert.csr`: cert used to generate the signed `aws_device_cert.pem`.

In the "Generating derived certificate" step, enter "y" to the questions "Sign the certificate" and "commit?".

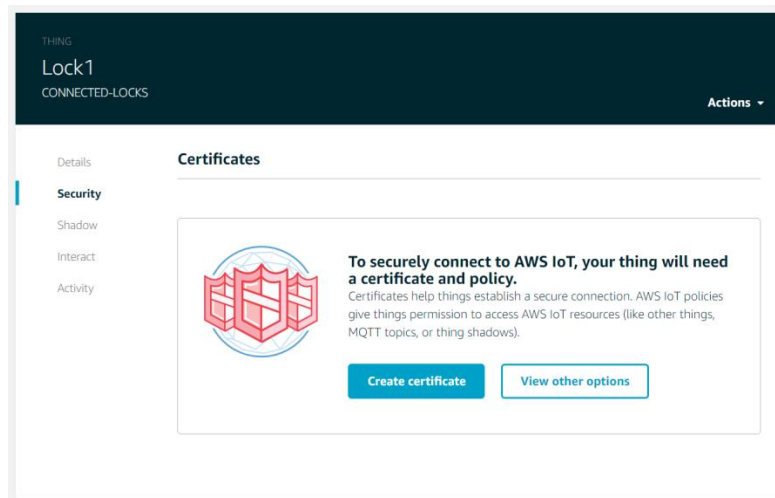
To comply with the TLS mutual authentication protocol, the Device Certificate, `aws_device_cert.pem`, is signed with the AWS Custom CA Key, `aws_custom_ca_key.pem`.

4.1.2. AWS Console and Board: Setup AWS IoT for the Example

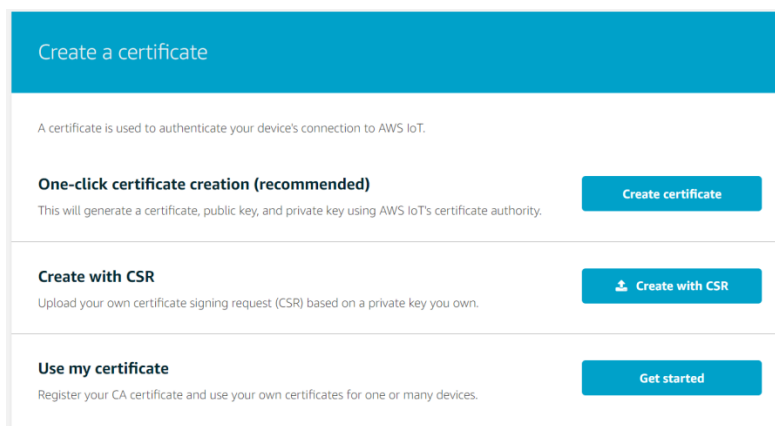
The user must go through the following steps to set and test the TLS connectivity with AWS IoT (step by step instructions are provided by the Amazon website (and also detailed below with screenshots): <http://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>):

1. Create an Amazon AWS account
2. Sign in to the AWS IoT Console
3. Create (Register) a “thing” in the Thing Registry
4. Register the CA to the AWS IoT.

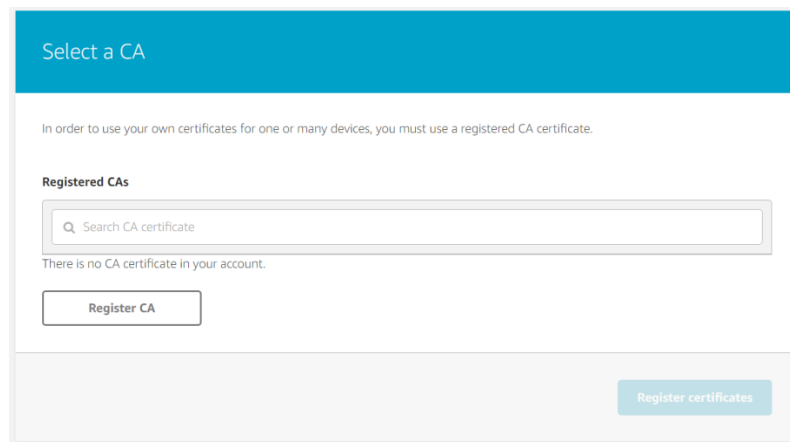
On the AWS console, in the “thing” details under “Security” the user has the option to create a certificate or chose other options. In the screenshots that follow we have created a “thing” called Lock 1 and we will continue the step by step instructions to register a CA and then the device certificate.



Under “View other Options” the user must choose “Use my certificate”.



Following the steps, the user must Register the CA first.



As per Amazon AWS,

to register your CA certificate, you must get a registration code from AWS IoT, sign a private key verification certificate with your CA certificate, and pass both your CA certificate and a private key verification certificate to the `register-ca-certificate` CLI command. The Common Name field in the private key verification certificate must be set to the registration code generated by the `get-registration-code` CLI command. A single registration code is generated per AWS account. You can use the `register-ca-certificate` command or the AWS IoT console to register CA certificates.

To register the CA certificate, follow the instructions in the screenshot below and check the two boxes in order to load and activate the CA.

In Step 4, "Use the CSR that was signed with the CA private key" use the AWS Custom CA Certificate and AWS Custom CA Key, i.e. `-CA aws_custom_ca_cert.pem -CAkey aws_custom_ca_key.pem`.

In Step 5, "Select CA certificate", upload the AWS Custom CA Certificate, i.e. `aws_custom_ca_cert.pem`.

Register a CA certificate

To use your own X.509 certificates, you must register a CA certificate with AWS IoT. You must prove you 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

```
1364cc4695ed6195cb3af2d2fa4d691f8898337c2845fddbf36de48a60c617
```

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

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgets Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []: 1364cc4695ed6195cb3af2d2fa4d691f8898337c2845fddbf36de48a60c617
Email Address []:
```

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 300 -sha256
```

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

Select CA certificate

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

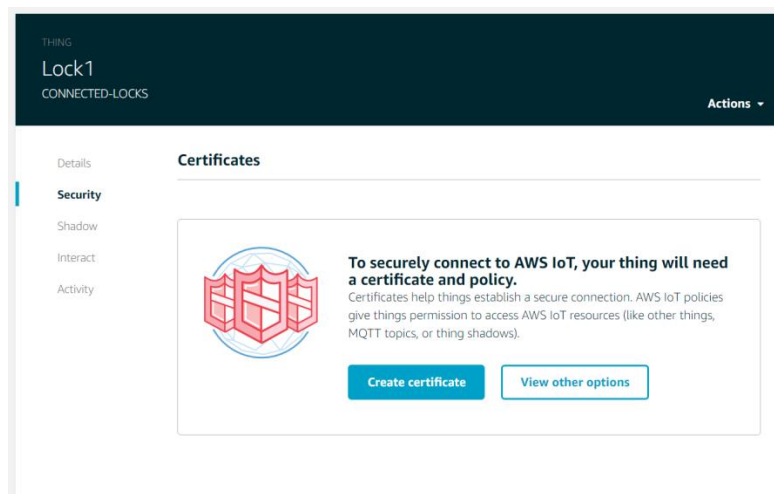
Select verification certificate

☒ Activate CA certificate

☒ Enable auto-registration of device certificates

5. Register the Device Certificate to the AWS IoT for the “thing”

On the AWS console, after the AWS Custom CA Certificate has been registered and activated, when going back to the “thing” that has been created, the user must click again on Security as in the screenshot below.



Then click on “View other options” and then “Use my certificate” (click on “Get started”).

Create a certificate

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

Create certificate

Create with CSR
Upload your own certificate signing request (CSR) based on a private key you own.

Create with CSR

Use my certificate
Register your CA certificate and use your own certificates for one or many devices.

Get started

As in the screenshot below, the user will need to click and select the CA that was just registered and then click on the bottom blue button “Register certificates”.

Select a CA

In order to use your own certificates for one or many devices, you must use a registered CA certificate.

Registered CAs

Search CA certificate

☐ d19542582dfb57fe285720b73e591f6f1cb6388420cf6434616f0fc9dc1127df

View

Register CA

Register certificates

As per the screen below the user has now the option to select to upload and register the Device Certificate.

Register existing device certificates

You can register device certificates signed by your CA certificate. Note that you must first register your CA certificate before uploading device certificates. You can upload up to 10 device certificates at a time.

Existing certificates

You have not selected any device certificates to upload yet.

Select certificates

Done

Register certificates

©2016-2018 Sequitur Labs, Inc. | PO Box 1127 | Issaquah, WA 98027 USA | p: +1 425 654 2048 | f: +1 425 654 2051 | www.sequiturlabs.com

15

Select the AWS Device Certificate, `aws_device_cert.pem` generated in 4.1.1, upload it to the AWS IoT “thing”, and press the Register certificate blue button (also check the “Activate all” radio button).

Register existing device certificates

You can register device certificates signed by your CA certificate. Note that you must first register your CA certificate before uploading device certificates. You can upload up to 10 device certificates at a time.

Existing certificates

	Deactivate all	Revoke all	
aws_device_cert.pem	<input checked="" type="radio"/>	<input type="radio"/>	Remove

Select certificates

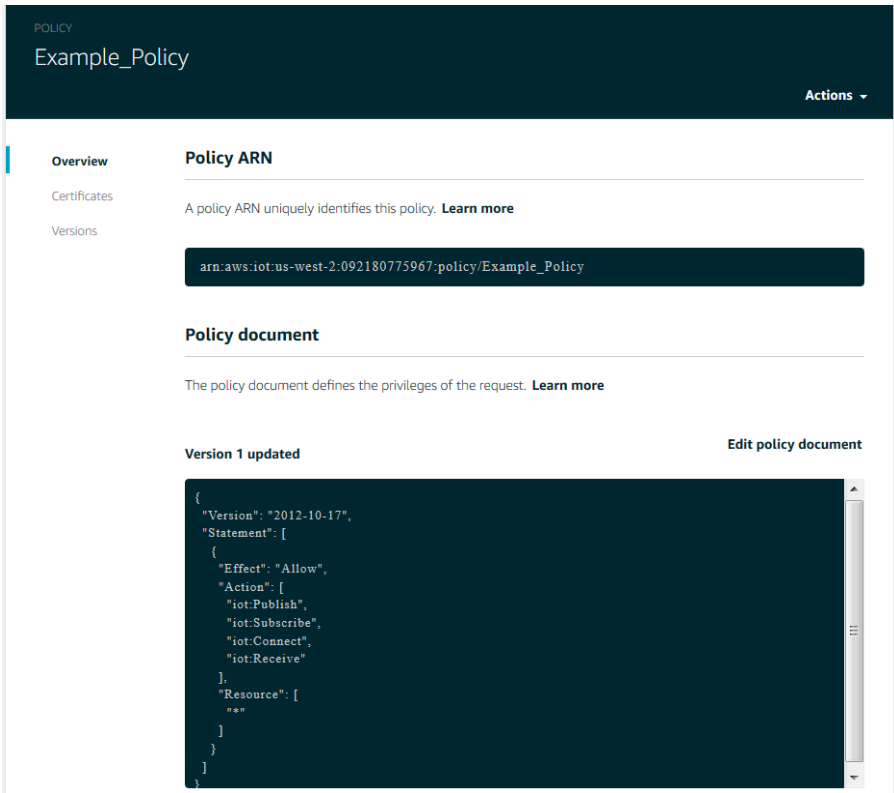
Done Register certificates

At this stage the AWS IoT cloud has a “thing” ready to allow a device to connect to it: the device is registered with an active certificate.

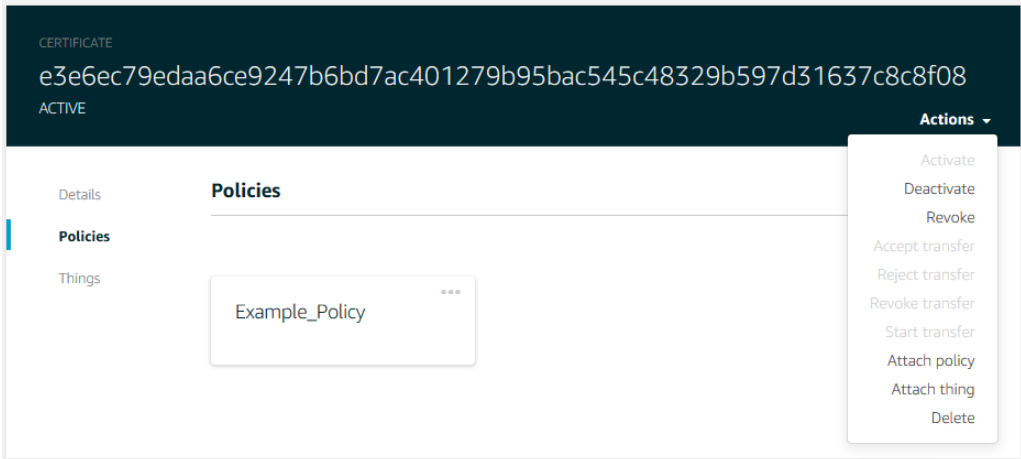
4.1.3. AWS Console: Create a Publish/Subscribe Policy

For this example, devices need to be attached to a policy that allows them to subscribe and publish. To accomplish this:

- Create a policy that allows subscription and publishing to a topic such as in the example policy shown in the image:
 - "iot:Publish"
 - "iot:Subscribe"



- Attach the device certificate to the policy, e.g.



4.1.4. AWS Console: Define the Topic to Subscribe

This example uses the AWS IoT MQTT client to subscribe to a topic to see messages from the device or to publish messages from the AWS IoT console on the topic.

As per Amazon AWS,

Devices publish messages on topics to which AWS IoT or your applications can respond. You can use the AWS IoT MQTT client to subscribe to these topics to see the content of these messages or publish messages on these topics to update device state.

AWS information on topics: <http://docs.aws.amazon.com/iot/latest/developerguide/topics.html>. There are reserved topics that are identified by "\$" symbol at the beginning. This example uses a reserved topic that has the following structure as defined by Amazon AWS:

Topic: \$aws/things/thingName/shadow/update/accepted

Allowed Operations: Subscribe

Description: The Thing Shadows service sends messages to this topic when an update is successfully made to a thing shadow. For more information see <http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html#update-accepted-pub-sub-topic>.

The topic for the example is defined as:

`$aws/things/seqlabs_IoTSS_thing/shadow/update/accepted`

where *thingName* is `seqlabs_IoTSS_thing`.

4.1.5. Linux Environment: Configure the AWS Example Application that Connects to AWS

This example application demonstrates the capabilities of the EmSPARK Suite for supporting integration with cloud services, in this case Amazon Web Services. The application uses the CoreLockr Crypto API and the CoreLockr Secure Certificates API and keys saved in the TEE.

This example shows how to use mbedTLS and the Suite to connect to an AWS server. It is intended for proof of concept only and is not intended for commercial use.

The "AWS IoT Embedded C Device SDK", "mbedTLS from ARM" was modified to use the CoreLockr Secure Certificate API for verifying certificates from AWS and to use the CoreLockr Cryptographic API for hashing and signing during the TLS handshake. See in [corelockr/examples/AWS/README.txt](#) the list of modifications to AWS SDK code.

The AWS SDK was originally downloaded from here:

https://s3.amazonaws.com/aws-iot-device-sdk-embedded-c/linux_mqtt_mbedtls-2.1.1.tar

Only the `subscribe_publish_sample` was modified to use the changes in the AWS SDK.

Software and Data Requirements

- AWS example application, [corelockr/examples/AWS](#)
- Completed the procedures described in the previous sections, starting at 4.1.1

Configuring the Application

The configuration file `'aws_iot_config.h'` in `corelockr/examples/AWS/linux_mqtt_mbedtls-2.1.1/samples/linux/subscribe_publish_sample/` must be updated based on the AWS 'Thing' that was created via the AWS IoT console:

- The `AWS_IOT_MQTT_HOST` must be set to the AWS URL configured Endpoint
- The `AWS_IOT_TOPIC` must be set to the topic to subscribe and publish defined in 4.1.3. This is the same topic used in the MQTT client of the AWS IoT console.

Building and Installing the Application

In Linux environment, change to `corelockr/examples/AWS/linux_mqtt_mbedtls-2.1.1/samples/linux/subscribe_publish_sample/`.

To compile the mbedTLS libraries which include the Sequitur Labs changes to use the Suite and build the `subscribe_publish_sample` executable, run `make`.

Transfer the executable to the board to a directory of your choice.

4.1.6. Board: Save the Certificates and Key in the TEE

Transfer to the board the `aws_management` tool located in `corelockr/examples/AWS/tools`. Transfer to the board the following certificates to be imported in the TEE:

- `aws_custom_ca_cert.pem`
- `aws_device_cert.pem`
- `aws_device_key.pem`

On the board, change to the directory where `aws_management` tool is located and execute it. The `aws_management` tool has three required parameters:

```
aws_management <aws_custom_ca_cert> <aws_device_cert> <aws_device_key>
```

To save the certs and key to the TEE execute:

```
./aws_management certs/aws_custom_ca_cert.pem  
certs/aws_device_cert.pem certs/aws_device_key.pem
```

This example command assumes that the key and cert files are under a `"certs"` directory.

After the tool execution, the following IDs for AWS are assigned:

- `aws_custom_ca_cert.pem`, is saved in the TEE as `com.seqlabs.aws.oem_root_cert`
- `aws_device_cert.pem`, is saved as `com.seqlabs.aws.device_cert`
- `aws_device_key.pem`, is saved as `com.seqlabs.aws.device_key`

4.1.7. Board: Execute the AWS Example Application

On the board, change to the directory where `subscribe_publish_sample` is located and execute the application. The execution output shows some of the steps of the TLS handshake and starts publishing values and subscribing to the topic set in `aws_iot_config.h`. The output includes:

```
AWS IoT SDK Version 2.1.1-  
Connecting...
```

```
Subscribing...
```

```
-->sleep
```

```
Subscribe callback
```

```
$aws/things/seqlabs_IoTSS_thing/shadow/update/accepted  hello from  
SeqLabs IoTSS SDK - QOS0 : 0
```

```
Subscribe callback
```

```
$aws/things/seqlabs_IoTSS_thing/shadow/update/accepted  hello from  
SeqLabs IoTSS SDK - QOS1 : 1
```

4.1.8. Board and AWS Console: Observe Published Messages

On the AWS console, select **Test**. On the **MQTT client** page, enter the topic for which the device is registered and select **Subscribe to topic**. In this example, the topic is:

```
$aws/things/seqlabs_IoTSS_thing/shadow/update/accepted
```

MQTT client ? Connected as iotconsole-1503007908750-0

Subscriptions

Subscribe to a topic

Publish to a topic

\$aws/things/seqlabs IoTSS_t... ✕

Subscribe

Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.

Subscription topic

\$aws/things/seqlabs IoTSS_thing/shadow/update/accepted

Subscribe to topic

Max message capture ?

100

Quality of Service ?

☒ 0 - This client will not acknowledge to the Device Gateway that messages are received

☐ 1 - This client will acknowledge to the Device Gateway that messages are received

MQTT payload display

☐ Auto-format JSON payloads (improves readability)

☒ Display payloads as strings (more accurate)

☐ Display raw payloads (in hexadecimal)

Observe the messages sent from the device

Subscriptions

Subscribe to a topic

Publish to a topic

\$aws/things/seqlabs IoTSS... ✕

\$aws/things/seqlabs IoTSS_thing/shadow/update/accepted

Export Clear Pause

Publish

Specify a topic and a message to publish with a QoS of 0.

\$aws/things/seqlabs IoTSS_thing/shadow/update/accepted

Publish to topic

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

hello from SeqLabs IoTSS SDK - QOS0 : 2

\$aws/things/seqlabs IoTSS_thing/shadow/upd... Aug 17, 2017 3:33:38 PM -0700 Exp... H...

hello from SeqLabs IoTSS SDK - QOS1 : 1

\$aws/things/seqlabs IoTSS_thing/shadow/upd... Aug 17, 2017 3:33:38 PM -0700 Exp... H...

hello from SeqLabs IoTSS SDK - QOS0 : 0

\$aws/things/seqlabs IoTSS_thing/shadow/upd... Aug 17, 2017 3:18:26 PM -0700 Exp... H...

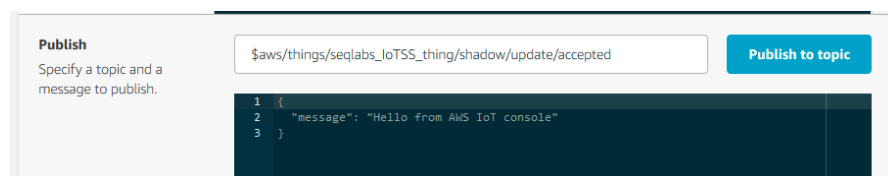
hello from SeqLabs IoTSS SDK - QOS1 : 67

\$aws/things/seqlabs IoTSS_thing/shadow/upd... Aug 17, 2017 3:18:26 PM -0700 Exp... H...

To send messages from the AWS console to the device, on the same page publish to the same topic configured on the device. Enter the topic on the **Publish** text box and select the **Publish to topic** button.

©2016-2018 Sequitur Labs, Inc. | PO Box 1127 | Issaquah, WA 98027 USA | p: +1 425 654 2048 | f: +1 425 654 2051 | www.sequiturlabs.com

21



On the board, observe the messages from the AWS console

```
-->sleep
Subscribe callback
$aws/things/seqlabs_IoTSS_thing/shadow/update/accepted {
  "message": "Hello from AWS IoT console"
}
```

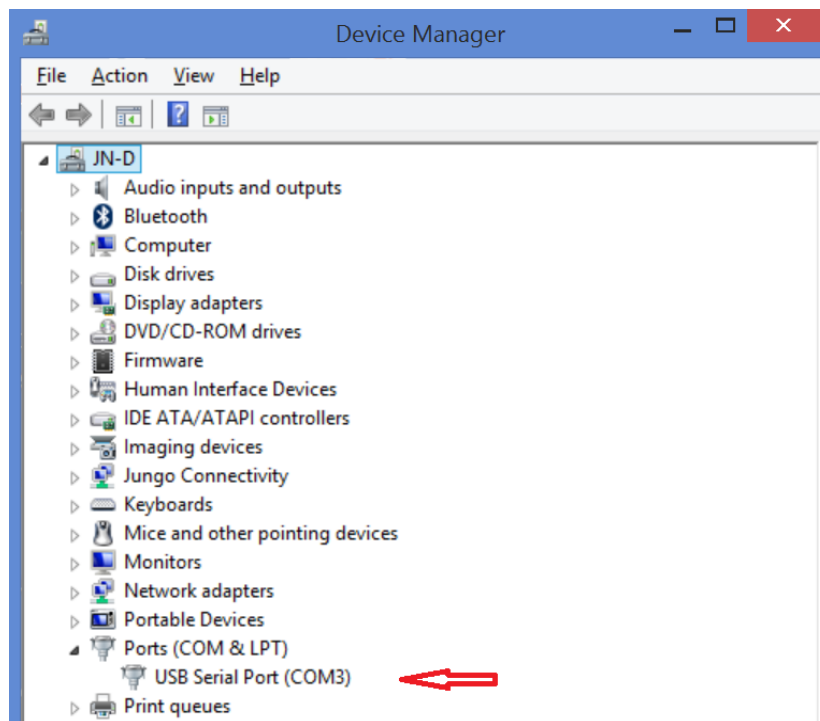
APPENDIX A: SECURE SAM-BA LOADER INSTRUCTIONS FOR WINDOWS

Board set up

- Remove power from the board and wait at least 30 seconds

Step 1

- Open Windows Device Manager.
- Connect a micro USB cable to J10 (debug) micro USB port on the board.
- Look on the Device Manager for the Serial COM port associated with this cable.
- This will be used to see the programming progress of the chip.



Step 2

- Open a serial communication program, such as "putty"
- Connect to the corresponding COM port with the following parameters
 - 115200 bps
 - No parity
 - 8 bits
 - 1 stop bit
 - No flow control
- 'RomBOOT' should appear on the serial terminal.



Step 3

- Connect a micro USB cable to the PC/power micro USB port on the board.
- This port will be used to program the board.
- **Step 4**
- Open a Windows command prompt and change to the Secure SAM-BA loader directory, `eval_kit_loader`
- Run the script to load the firmware to the board, and follow its instructions:

```
> flash_eval_kit.bat
```

The script will identify the correct port attached to the host computer and start the flashing process.

```
C:\WINDOWS\system32\cmd.exe

S:\eval_kit_loader>flash_eval_kit.bat
S:\eval_kit_loader>set SAMBA_DIR=sam-ba_3.3_win32
S:\eval_kit_loader>sam-ba_3.3_win32\sam-ba.exe -x eval_kit.qml
Opening serial port 'COM4'
Connection opened.
Detected memory size is 134217728 bytes.
Page size is 256 bytes.
Buffer is 89600 bytes (350 pages) at address 0x0022a160.
Supported erase block sizes: 4KB, 32KB, 64KB
Erased 65536 bytes at address 0x00000000 (0.52%)
Erased 65536 bytes at address 0x00010000 (1.04%)
Erased 65536 bytes at address 0x00020000 (1.56%)
Erased 65536 bytes at address 0x00030000 (2.08%)
Erased 65536 bytes at address 0x00040000 (2.60%)
Erased 65536 bytes at address 0x00050000 (3.13%)
Erased 65536 bytes at address 0x00060000 (3.65%)
```

Step 5

- When instructed on the Windows terminal, reset the board (do not power it off, use the NRST button to reset).


```

Wrote 89600 bytes at address 0x004b3200 (95.70%)
Wrote 89600 bytes at address 0x004c9000 (97.41%)
Wrote 89600 bytes at address 0x004dee00 (99.12%)
Wrote 46336 bytes at address 0x004f4c00 (100.00%)
-- new boot config --
BSCR=0x00000004 / BUREG0,VALID
BUREG0=0x20060004 / QSPI0_IOSET1,QSPI1_IOSET2,SPI0_IOSET1,SPI1_IOSET1,NFC_IOSET1,
MODE_CM4C
BUREG1=0x00000000 / QSPI0_IOSET1,QSPI1_IOSET1,SPI0_IOSET1,SPI1_IOSET1,NFC_IOSET1,
BUREG2=0x00000000 / QSPI0_IOSET1,QSPI1_IOSET1,SPI0_IOSET1,SPI1_IOSET1,NFC_IOSET1,
BUREG3=0x00000000 / QSPI0_IOSET1,QSPI1_IOSET1,SPI0_IOSET1,SPI1_IOSET1,NFC_IOSET1,
FUSE=0x00000000 / QSPI0_IOSET1,QSPI1_IOSET1,SPI0_IOSET1,SPI1_IOSET1,NFC_IOSET1,
Connection closed.
"Reset the board, then press any key to continue..."
Press any key to continue . . .

```

- 'Secure Boot Mode' will be displayed on the serial terminal.



```

RomBOOT
Secure Boot Mode

```

- On the Windows command line, press any key to write the customer key
- The programming process on the Windows command line will complete.

```

Connection closed.
"Reset the board, then press any key to continue..."
Press any key to continue . . .

"Writing Customer Key..."

S:\eval_kit_loader>sam-ba_3.3_win32\sam-ba.exe -p secure -d sama5d2 -m write_cust
Opening secure port 'COM4'
Connection opened.
Connection closed.

"=====
"= >>>>>>          DONE          <<<<<< ="
"= >>>>>>    RESET BOARD TO BOOT  <<<<<< ="
"=====

S:\eval_kit_loader>

```

Step 6

- Reset the board once more to start provisioning the board.
- The serial terminal will print the provisioning messages.

```

RomBOOT
Secure Boot Mode

CoreBOOT Secure Provisioning - seqlabs_1.2.0-2.3.0 [HD] 3.8-alpha7-g89cee86 (Thu
:47:38 PST 2019)

Commencing Provisioning
Do not reset the board
Creating Device Key and Certificate
Re-encrypting payloads with randomly generated device keys
Checking Media For CoreTEE, DTB, Linux, Ramdisk
Update_address: 4952064 / 0x4b9000 backup_address: 0 / 0x0
Ramdisk empty [zero size], will not be used.

Checking TEEBOOT [0x7eb0]
Process Payload: 1 of 5
..... 100%

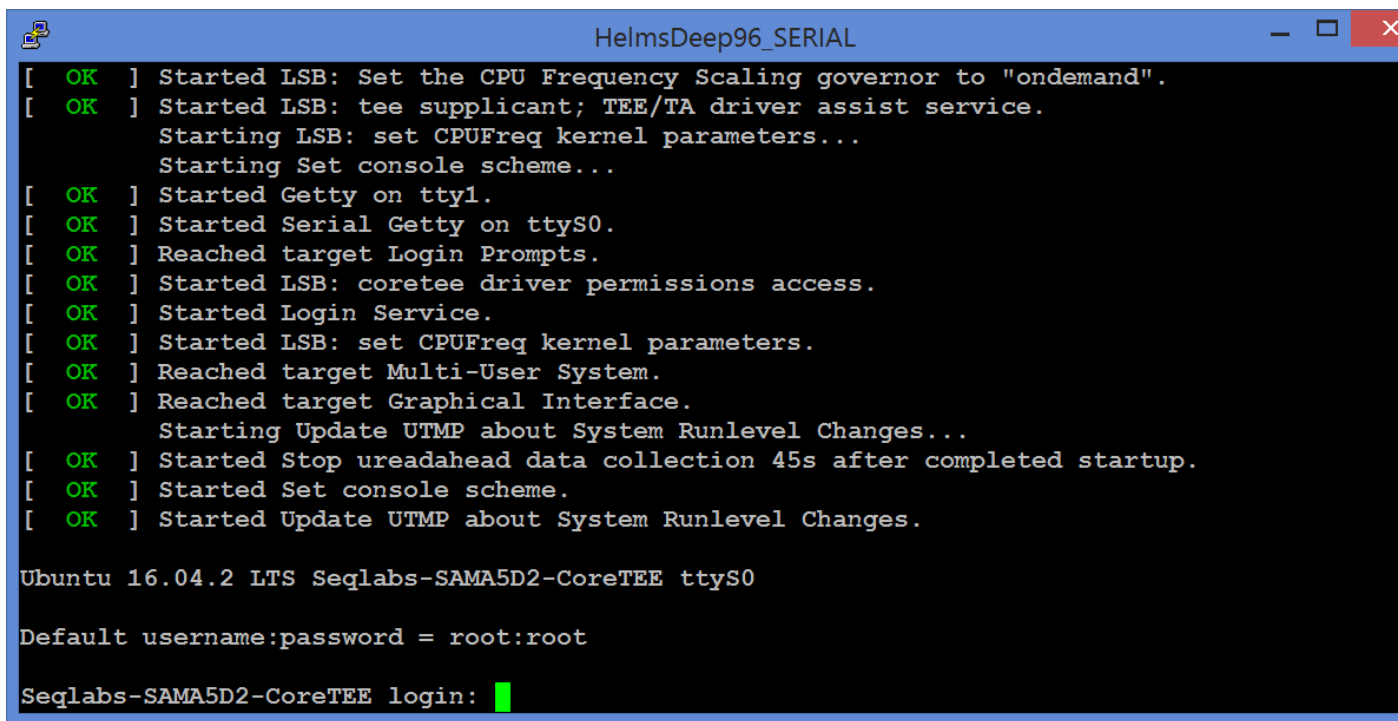
Checking CoreTEE [0x82d10]
Process Payload: 2 of 5
..... 48%
..... 97%
... 100%

Checking Linux [0x3db190]
Process Payload: 3 of 5
..... 6%
..... 12%

```

Step 7

- Insert the SD card with the file system and press the "RESET" button.
- The board will begin booting.



```

[ OK ] Started LSB: Set the CPU Frequency Scaling governor to "ondemand".
[ OK ] Started LSB: tee supplicant; TEE/TA driver assist service.
Starting LSB: set CPUFreq kernel parameters...
Starting Set console scheme...
[ OK ] Started Getty on tty1.
[ OK ] Started Serial Getty on ttyS0.
[ OK ] Reached target Login Prompts.
[ OK ] Started LSB: coretee driver permissions access.
[ OK ] Started Login Service.
[ OK ] Started LSB: set CPUFreq kernel parameters.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Stop ureadahead data collection 45s after completed startup.
[ OK ] Started Set console scheme.
[ OK ] Started Update UTMP about System Runlevel Changes.

Ubuntu 16.04.2 LTS Seqlabs-SAMA5D2-CoreTEE ttyS0

Default username:password = root:root

Seqlabs-SAMA5D2-CoreTEE login: █

```

NOTE:

To reset the board back to a mode where it can be reflashed, remove all power to the board for 30 seconds.

APPENDIX B: SECURE SAM-BA LOADER INSTRUCTIONS FOR LINUX

1. Board set up. Remove power from the board and wait at least 30 seconds.
2. On the Linux host computer, make sure the user has permissions to access `/dev/ttyACM0` and `/dev/ttyUSB0`.
3. Use a serial communication program to observe the Secure World console. For example, open "minicom". Connect the serial terminal with the following parameters:
 - 115200 bps
 - No parity
 - 8 bits
 - 1 stop bit
 - No flow control

'RomBOOT' will appear on the serial terminal.

4. Connect a micro USB cable to the PC/power micro USB port on the board. This port will be used to program the board.
5. Open a second terminal to program the board and change to the `eval_kit_loader` directory. Run the script to load the firmware to the board, and follow its instructions.

```
$ ./flash_eval_kit.sh
```

6. When instructed, reset the board (do not power it off, use the NRST button to reset).

'Secure Boot Mode' will be displayed on the serial terminal.

7. On the Linux terminal, press any key to write the customer key.

The programming process on the Linux command line will complete.

```
Connection closed.
Reset the board, then press any key to continue...
Writing Customer Key...
Opening secure port 'ttyACM0'
Connection opened.
Connection closed.
```

```
=====
= >>>>>                DONE                <<<<<< =
= >>>>>          RESET BOARD TO BOOT          <<<<<< =
=====
```

8. Reset the board once more to start provisioning the board.

The serial terminal will print the provisioning messages, e.g.

```
CoreBOOT Secure Provisioning - seqlabs_1.2.0-2.3.0 [HD] 3.8-alpha7-  
g89cee86 (Tue Nov 12 23:27:36 PST 2019)
```

```
Commencing Provisioning  
Do not reset the board  
Creating Device Key and Certificate  
Re-encrypting payloads with randomly generated device keys  
Checking Media For CoreTEE, DTB, Linux, Ramdisk
```

9. Insert the SD card with the file system and press the "RESET" button.

The board will begin booting.

NOTE:

To reset the board back to a mode where it can be reflashed, remove all power to the board for 30 seconds.

CHANGE HISTORY

DATE	VERSION	RESPONSIBLE	DESCRIPTION
December 13, 2019	1.0	Julia Narvaez	Produced document for release.