Amazon FreeRTOS Qualification Developer Guide

Document Version V1.1.6

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon FreeRTOS Qualification: Developer Guide

Copyright © 2018, 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Revision History

Date	Version	Change History	Compatible Amazon FreeRTOS Version
July 31, 2018	1.0.0	Initial Version.	1.3.0
August 09, 2018	1.0.1	Updates in appendices: • Updates in Porting Order chart. • Updates in PKCS#11 “Porting” section. • File path changes in TLS “Test Setup” section and TLS Server Setup step 9. • Fix hyperlinks in MQTT prerequisite section. • Add AWS CLI config instruction in BYOC certificate creation example.	1.3.1 1.3.2
August 27, 2018	1.1.0	Added tests for Over-the-air (OTA) updates and guidelines for bootloader	1.4.0 1.4.1
October 08, 2018	1.1.1	Added clarifications in (B3.1) tests enable constants table. Updated path changes of Unity module in “Create Test Project (B2.2)” section. Updates in appendices: • Updated Porting Order Flowchart. • Updated client credential certificate/key variable names in TLS “Test Setup” section. • Corrections of a file path in Secure Socket “Test Setup” section and TLS Server Setup. • Updated the path for echo servers in TLS “Test Setup” section and TLS Server Setup.	1.4.2
November 07, 2018	1.1.2	Updates in appendices: • Modifications of PKCS #11 PAL layer interface changes in PKCS #11 porting section. • Updated GSG template, “Download and Configure Amazon FreeRTOS” section. • Updated the path of CertificateConfigurator.html.	1.4.3 1.4.4
November 26, 2018	1.1.3	Added BLE section. Updated with IoT Device Tester information. Added cmake preparation section.	To be assigned
December 13, 2018	1.1.4	Updates in appendices: • Added lwIP porting support.	1.4.5
December 27, 2018	1.1.5	Updated “Checklist for Qualification” appendix. Cmake files are required for qualification.	1.4.5 1.4.6

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

February 25, 2019	1.1.6	Updated "Getting Started Guide Template" appendix.	1.4.5 1.4.6 1.4.7

Contents

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

1 Introduction

1.1 What is the Amazon FreeRTOS Qualification?

The Amazon FreeRTOS Qualification (AFQ) defines a process that the author of an Amazon FreeRTOS port[1] must follow, and a set of tests that the port must pass, in order for the port to be described as 'qualified by Amazon'. Amazon only distributes and supports Amazon FreeRTOS ports that have passed the qualification program. The purpose of the AFQ is to give developers confidence that qualified Amazon FreeRTOS ports behave correctly and consistently with each other.

AWS provides a free test automation framework called AWS IoT Device Tester for Amazon FreeRTOS, which developers can use to automate running of the qualification tests. See https://aws.amazon.com/freertos/device-tester for more information on how to set it up.

1.2 Target System Requirements

It is recommended that Amazon FreeRTOS is only qualified on microcontrollers (MCUs) that have a minimum processing speed of 25MHz, a minimum of 64K bytes of RAM, and a minimum of 128K bytes of program memory per executable image stored on the MCU. For future qualification requirement with Over-the-air update (OTA) functionality, two executable images must be stored in program memory at the same time.

1.3 Document Outline

This document guides you through setting up your Amazon FreeRTOS project, porting Amazon FreeRTOS libraries, building and testing your ports using the Amazon FreeRTOS Qualification tests. Appendixes provide more detail information on each Amazon FreeRTOS library.

We also prepared a checklist for the process: Appendix O: Checklist for Qualification

[1] An Amazon FreeRTOS port is a board-specific implementation of APIs for certain Amazon FreeRTOS libraries. The port enables these APIs to work on the specific board, and implements the required integration with device drivers and BSPs provided by the platform vendor. It should also include any configuration adjustments (e.g. clock rate, stack size, heap size etc.) required by the board.

2 Amazon FreeRTOS Qualification Test Project

Follow the steps below.

2.1 Download Amazon FreeRTOS Source Code

You can download the Amazon FreeRTOS source code and test code from GitHub:

https://github.com/aws/amazon-freertos

Please download the latest release code from the "release" branch. You should import the repository to your own private GitHub repository and configure to watch Amazon FreeRTOS public repository. You will get notifications if there are new releases on our release branch.

If you are using Windows, you must keep the file path short (for example clone to C:\AFreeRTOS rather than C:\Users\username\programs\projects\AmazonFreeRTOS\) to avoid a Windows limitation with long file paths. The chosen folder will be referred as **$AFR_HOME** from here on in the document.

2.2 Set Up Your Amazon FreeRTOS Project

At the end of this step, you will have a working project that can write to a serial console.

2.2.1 Preparing Amazon FreeRTOS Folders

All qualified Amazon FreeRTOS ports use the same directory structure. New files, including IDE project files, must be created in the correct folder locations.

The directory structure is explained below. Detailed instructions on how to create the same directory structure are listed in the grey box.

Directory Structure:

The three root level folders under $AFR_HOME are:

```
$AFR_HOME
├───demos              Contains projects that build demo applications
├───lib                Contains Amazon FreeRTOS and third-party libraries
└───tests              Contains projects that build qualification tests
```

Create your project in the `tests` folder, which is structured as follows:

```
$AFR_HOME
└───tests
    ├───common         Contains files built by all test projects
    ├───pc             Contains a reference test project for the FreeRTOS Windows port
    └───vendor         Template, to be renamed to the name of the MCU vendor
        └───board      Template, to be renamed to the name of the development board
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Instructions:

1. Rename the **$AFR_HOME/tests/vendor** folder to the name of the company that manufactures the MCU – from here on the folder is referred to as [vendor].
2. Rename the **$AFR_HOME/tests/board** folder to the name of the development board being qualified – from here on the folder is referred to as [board].

The **$AFR_HOME/tests/[vendor]/[board]** folder is a template provided to simplify the creation of a new test project and ensures all test projects have a consistent organization. It has the following structure:

```
$AFR_HOME
└───tests
    └───[vendor]
        └───[board]
            ├───common
            │   ├───application_code   Contains main.c, which itself contains main()
            │   │   └───vendor_code    Contains vendor supplied board specific files
            │   └───config_files       Contains Amazon FreeRTOS config files
            └───ide                    Contains an IDE specific project
```

Test projects always require vendor-supplied driver libraries. Some vendor-supplied libraries, such as a header file that maps a GPIO output to an LED output, are specific to the target development board. Other vendor-supplied libraries, such as the GPIO library itself, are specific to the target MCU family.

For Vendor-supplied driver libraries that are **specific to the target development board**:

Instructions (continued):

3. Save any required vendor-supplied libraries that are specific to the board in the **$AFR_HOME/tests/[vendor]/[board]/common/application_code/vendor_code** folder.
4. Rename the **$AFR_HOME/tests/[vendor]/[board]/ide** folder to the name of the IDE that will be used to build the test project – from here on the folder is referenced as [IDE].

Vendor-supplied driver libraries that are **specific to the target MCU family** belong in the **$AFR_HOME/lib/third_party/mcu_vendor** folder, which has the following structure:

```
$AFR_HOME
└─lib
   ├──AmazonLib1      Contains an Amazon FreeRTOS library (example name only)
   ├──AmazonLib2      Contains an Amazon FreeRTOS library (example name only)
   └──third_party     Contains all non-board specific third-party libraries
      ├───Lib1        Contains a third-party library (example name only)
      ├───Lib2        Contains a third-party library (example name only)
      └───mcu_vendor  Contains vendor-supplied MCU specific libraries
          └──vendor   Template, to be renamed to the name of the MCU vendor
             └─driver_library Template, to be renamed to the library name
                └─driver_library_version Template, to be renamed to the library version
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Instructions (continued):

5. Rename the **$AFR_HOME/lib/third_party/mcu_vendor/vendor** folder to [vendor] (the name of the company that manufactures the MCU).
6. Likewise, rename the contained **driver_library** folder to the name of the vendor's MCU specific driver library, and the contained **driver_library_version** folder to the version number of the vendor's MCU specific driver library.
7. Copy the vendor-supplied driver library into the newly renamed **driver_library_version** folder.

NOTE: DO NOT save vendor-supplied libraries that are specific to the MCU family anywhere within either the **$AFR_HOME/tests** or **$AFR_HOME/demos** folders!

2.2.2 Create the Test Project

All qualified Amazon FreeRTOS test projects look the same when viewed from within an IDE. This section describes and demonstrates the required project structure. By the end of this section you will have a project with FreeRTOS Kernel libraries ready to run. The next section (2.32.3) will cover porting of other Amazon FreeRTOS libraries into the project.

Instructions:

1. Read this section of the document, and then **replicate** the project it describes, but using the selected IDE, and targeting the hardware being qualified. Take care to ensure the structure of the created project matches that described below.

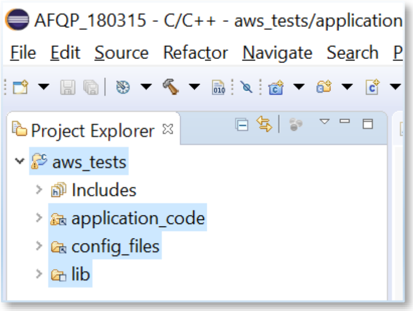
NOTE 1: All files in the project must be built in the file's original position within the folder structure. They are imported into the project by linking the files. Never directly copy files into the project's folder or use absolute file paths.

NOTE 2: If you are using an Eclipse based IDE, do not configure the project to build all the files in any given folder. Instead, add source files into the project by linking to each source file individually.

The project is called **aws_tests**. Under **aws_tests**, there are three virtual folders. In this context, a virtual folder is created in an IDE to better organize the source code. It may not correspond to a physical directory on disk. The three virtual folders are **application_code**, **config_files** and **lib**, as described below:

```
aws_tests                   The project name
├──application_code           Contains application logic, in this case it is AFQ test code
├──config_files               Contains header files that configure Amazon FreeRTOS libraries
└──lib                        Contains Amazon and third-party libraries
```

Error! Reference source not found. shows how the top three virtual folders appear in an IDE. The depicted IDE is Eclipse, but the structure is the same in all IDEs.

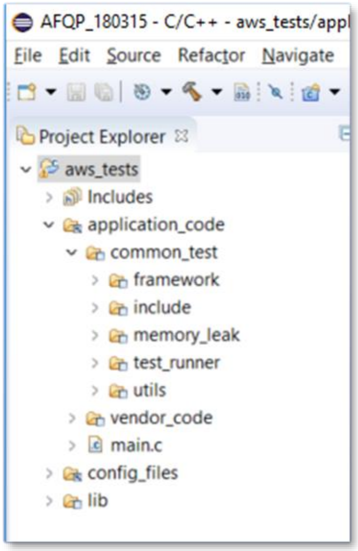

AFQP_180315 - C/C++ - aws_tests/application
File Edit Source Refactor Navigate Search P
Project Explorer
aws_tests
Includes
application_code
config_files
lib

NOTE:

- The “Includes” folder is generated automatically by Eclipse. It is not part of the required structure

Figure 1 The top three virtual folders viewed in an IDE’s project explorer view

Figure 2 shows the contents of the **application_code** virtual folder.



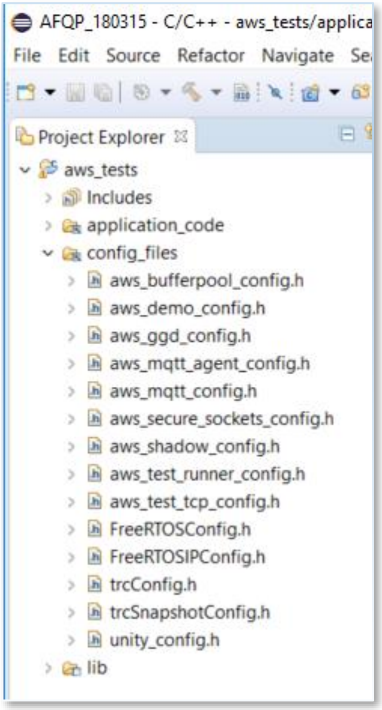
NOTE:

- The **main.c** file, and the **vendor_code** folder, are (physically on the disk) located in the **$AFR_HOME/tests/[vendor]/[board]/common/ application_code** folder.
- **common_test** is a virtual folder – it does not actually exist within the **$AFR_HOME** directory structure. The folders under **common_test** are located in the **$AFR_HOME/tests/common** folder. The project builds the source files located in those folders.

Figure 2 An IDE project with the application_code virtual folder expanded

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Figure 3 shows the contents of the **config_files** virtual folder.


AFQP_180315 - C/C++ - aws_tests/applica
File Edit Source Refactor Navigate Se
Project Explorer

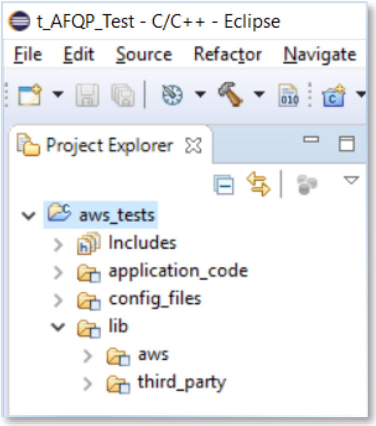
```
aws_tests
  Includes
  application_code
  config_files
    aws_bufferpool_config.h
    aws_demo_config.h
    aws_ggd_config.h
    aws_mqtt_agent_config.h
    aws_mqtt_config.h
    aws_secure_sockets_config.h
    aws_shadow_config.h
    aws_test_runner_config.h
    aws_test_tcp_config.h
    FreeRTOSConfig.h
    FreeRTOSIPConfig.h
    trcConfig.h
    trcSnapshotConfig.h
    unity_config.h
  lib
```

NOTE:

- The files shown under **config_files** are located in the **$AFR_HOME/tests/[vendor]/[board]/common/config_files** folder.

Figure 3 An IDE project with the config_files folder expanded

Figure 4 shows the contents of the **lib** virtual folder.


t_AFQP_Test - C/C++ - Eclipse
File Edit Source Refactor Navigate
Project Explorer

```
aws_tests
  Includes
  application_code
  config_files
  lib
    aws
    third_party
```

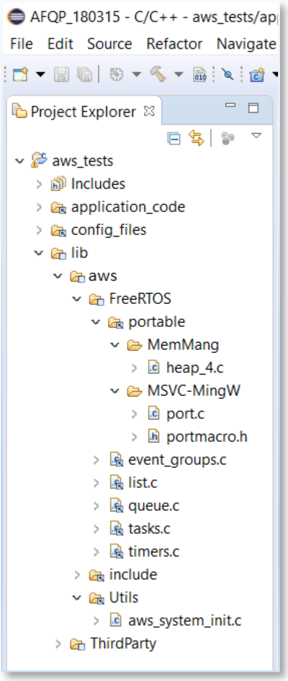
NOTE:

- **aws** and **third_party** are virtual folders.

Figure 4 IDE project with the lib group expanded

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Figure 5 shows the contents of the **lib/aws** folder, which contains AWS (as opposed to third party) provided libraries. It only contains FreeRTOS kernel library at this stage. You will import more libraries in this folder during later porting effort.


AFQP_180315 - C/C++ - aws_tests/ap
File Edit Source Refactor Navigate
Project Explorer

```
aws_tests
  Includes
  application_code
  config_files
  lib
    aws
      FreeRTOS
        portable
          MemMang
            heap_4.c
          MSVC-MingW
            port.c
            portmacro.h
        event_groups.c
        list.c
        queue.c
        tasks.c
        timers.c
      include
      Utils
        aws_system_init.c
    ThirdParty
```

Figure 5 IDE project with the lib/AWS group expanded

NOTE:

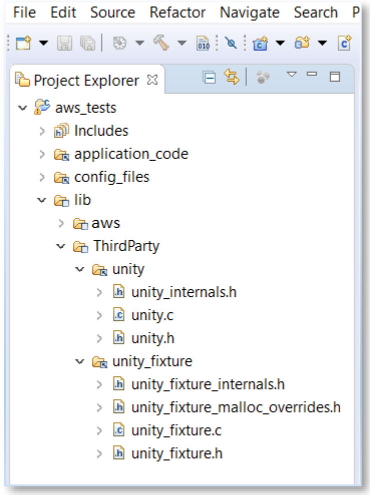
- The files and folders shown under **lib/aws/FreeRTOS** are located in the **$AFR_HOME/lib/FreeRTOS** folder. The figure shows **lib/aws/FreeRTOS/portable/MSVC-MingW** being included in the project. That folder contains the FreeRTOS kernel Windows port and should be substituted with whichever folder contains the correct FreeRTOS port for your target IDE and MCU, see below instruction.
- The file shown under **lib/aws/FreeRTOS/portable/MemMang** is located in the **$AFR_HOME/lib/FreeRTOS/MemMang** folder. It is FreeRTOS memory management implementation.
- The files shown under **lib/aws/include** are located in the **$AFR_HOME/lib/include** folder. Although not shown in Figure , it includes all the header files and folders under **$AFR_HOME/lib/include**.
- **aws_system_init.c** is located in the **$AFR_HOME/lib/utils** folder.

Instructions (continued):

1. Replicate the folder structure (continued)
   a. Import the FreeRTOS Kernel port for your compiler and architecture **in place of** *lib/aws/FreeRTOS/portable/MSVC_MingW in Figure 5.* **$AFR_HOME/lib/FreeRTOS/portable** contains the FreeRTOS kernel port files organized first by compiler, and then by architecture.
   b. Import one of the FreeRTOS Kernel memory management implementation to **lib/aws/FreeRTOS/portable/MemMang**. For Amazon FreeRTOS, we use heap_4.c. For more information, please visit FreeRTOS Memory Management

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Figure 6 shows the contents of the **lib/third_party** directory.



```
aws_tests
  Includes
  application_code
  config_files
  lib
    aws
    ThirdParty
      unity
        unity_internals.h
        unity.c
        unity.h
      unity_fixture
        unity_fixture_internals.h
        unity_fixture_malloc_overrides.h
        unity_fixture.c
        unity_fixture.h
```

Figure 6 IDE project with the lib/third_party group expanded

NOTE:

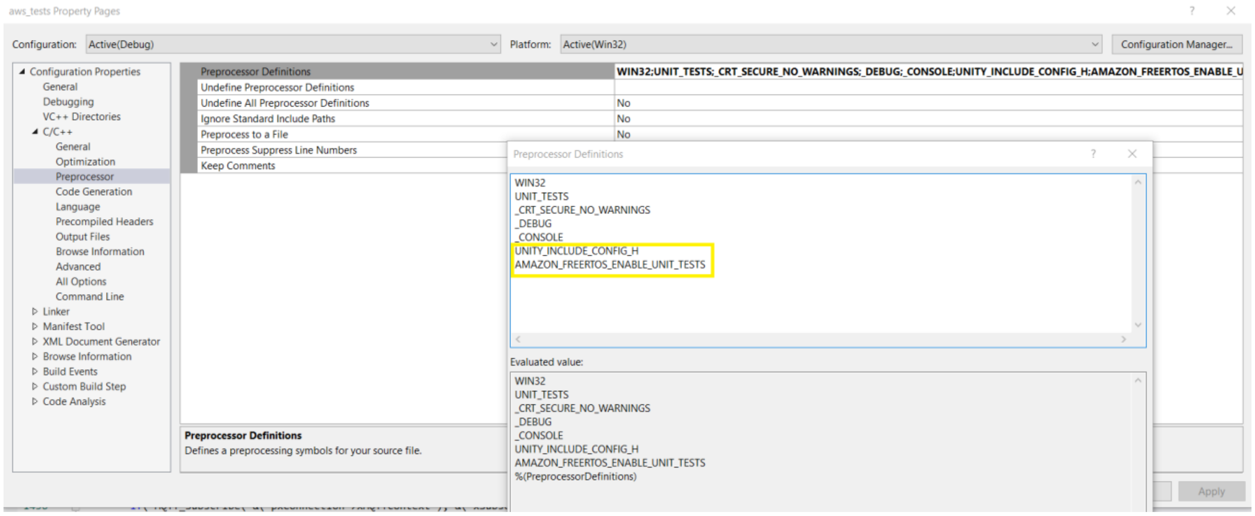
- **unity** and **unity_fixture** are virtual folders.
- The files shown under **unity** are located in the **$AFR_HOME/lib/third_party/unity/src** folder.
- The files shown under **unity_fixture** are located in the **$AFR_HOME/lib/third_party/unity/extras/fixture** folder.
- Although not shown in Figure , also add the MCU specific vendor-supplied driver libraries that were saved in the **$AFR_HOME/lib/third_party/[mcu_vendor]/[vendor]/[driver_library]/[driver_library_version]** folder (see section Preparing Amazon FreeRTOS Folders).

Instructions (continued, to be followed after creating the project):

2. Make sure the following compiler include paths are set in the project property.
   a. **$AFR_HOME/tests/common/include**, which is aws_tests/application_code/common_tests/include when viewed in the IDE project.
   b. **$AFR_HOME/lib/include**, which is aws_tests/lib/aws/include when viewed in the IDE.
   c. **$AFR_HOME/lib/include/private**, which is aws_tests/lib/aws/include/private when viewed in the IDE.
   d. **$AFR_HOME/lib/FreeRTOS/portable/[compiler]/[architecture]**, which is aws_tests/lib/aws/FreeRTOS/portable/[compiler]/[architecture] when viewed in the IDE.
   e. **$AFR_HOME/lib/third_party/unity/src**, which is aws_tests/lib/third_party/unity when viewed in the IDE.
   f. **$AFR_HOME/lib/third_party/unity/extras/fixture/src**, which is aws_tests/lib/third_party/unity_fixture when viewed in the IDE.
   g. **$AFR_HOME/demos/vendor/board/common/config_files**, which is aws_tests/config_files when viewed in the IDE.
   h. Any paths necessitated by vendor-supplied driver libraries.

3. Define the following two project level macros in your IDE:
   - UNITY_INCLUDE_CONFIG_H
   - AMAZON_FREERTOS_ENABLE_UNIT_TESTS

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.



Configuration: Active(Debug)
Platform: Active(Win32)

Preprocessor Definitions	WIN32;UNIT_TESTS;_CRT_SECURE_NO_WARNINGS;_DEBUG;_CONSOLE;UNITY_INCLUDE_CONFIG_H;AMAZON_FREERTOS_ENABLE_U
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	
Keep Comments	

Preprocessor Definitions
Defines a preprocessing symbols for your source file.

Preprocessor Definitions

```
WIN32
UNIT_TESTS
_CRT_SECURE_NO_WARNINGS
_DEBUG
_CONSOLE
UNITY_INCLUDE_CONFIG_H
AMAZON_FREERTOS_ENABLE_UNIT_TESTS
```

Evaluated value:

```
WIN32
UNIT_TESTS
_CRT_SECURE_NO_WARNINGS
_DEBUG
_CONSOLE
UNITY_INCLUDE_CONFIG_H
AMAZON_FREERTOS_ENABLE_UNIT_TESTS
%(PreprocessorDefinitions)
```

Figure 7 Preprocessor Macro Definitions Example (Visual Studio)

Visual Studio 2017 Example: Project Properties => Preprocessor => Preprocessor Definitions

4. Implement configPRINT_STRING() as described in Appendix A.
5. Make sure the new project builds successfully, that the resultant executable binary can be loaded to the target hardware. If you run the project in debug mode, the pc should stop at the first line of `main()`.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

2.3 Port, Build and Test Libraries and Demos

2.3.1 Port Libraries, Build and Test Libraries

With the folder structure and test projects prepared you are ready to start porting and testing the Amazon FreeRTOS libraries. You must enable **AFQ test group** first.

$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h contains a macro defined shown below. Uncomment the following line:

```
/*#define testrunnerAFQP_ENABLED */
```

Then bring an Amazon FreeRTOS library into your test project and port the library to your hardware. The libraries are listed in below table. The detailed instruction on porting and testing procedures are listed in the appendices – one appendix per library. The order of the appendices accounts for interdependencies between libraries so should be followed in turn.

Library	Details in Location	Notes
configPRINT_STRING Macro	Appendix A	Required for console output.
FreeRTOS Kernel	Appendix B	Required for the FreeRTOS kernel.
Wi-Fi Management	Appendix C	Required only if hardware supports network connectivity over Wi-Fi.
FreeRTOS TCP/IP Stack	Appendix D	Required only if a board does not have its own TCPIP stack support.
Secure Sockets	Appendix E	Required for AWS cloud connectivity.
PKCS#11	Appendix F	Required for over-the-air (OTA) and TLS support.
TLS	Appendix G	Required for TLS support.
MQTT	Appendix H	Required for AWS cloud connectivity.
OTA Updates	Appendix I	Required for OTA updates.
Bootloader (demo)	Appendix J	Required for OTA updates.
Bluetooth Low Energy (Beta)	Appendix K	Required for Bluetooth Low Energy (BLE) support

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

There are constants defined in **$AFR_HOME/tests/[vendor]/[board]/common/config_files/*aws_test_runner_config.h*** file which can be used as a switch to trigger a test set for a library. To enable a set of tests, set the constant to 1. These constants listed in table below:

Defined constant to trigger a set of tests on a library	Default value. (set to '1' for enabling the test)	Required for AFQ
testrunnerFULL_CBOR_ENABLED	0	
testrunnerFULL_OTA_AGENT_ENABLED	0	Y (if supports OTA)
testrunnerFULL_OTA_PAL_ENABLED	0	Y (if supports OTA)
testrunnerFULL_MQTT_ALPN_ENABLED	0	
testrunnerFULL_MQTT_STRESS_TEST_ENABLED	0	
testrunnerFULL_MQTT_AGENT_ENABLED	0	
testrunnerFULL_TCP_ENABLED	0	Y
testrunnerFULL_GGD_ENABLED	0	
testrunnerFULL_GGD_HELPER_ENABLED	0	
testrunnerFULL_SHADOW_ENABLED	0	
testrunnerFULL_MQTT_ENABLED	0	Y
testrunnerFULL_PKCS11_ENABLED	0	Y
testrunnerFULL_CRYPTO_ENABLED	0	
testrunnerFULL_TLS_ENABLED	0	Y
testrunnerFULL_WIFI_ENABLED	0	Y
testrunnerFULL_BLE_ENABLED	0	Y (if supports BLE)

This is a vendor configurable file. If your board does not support a certain feature (i.e. OTA), you can define the corresponding test sets to not supported, like the following:

```
#define testrunnerFULL_OTA_AGENT_ENABLED      testrunnerUNSUPPORTED
```

NOTE: If you have setup AWS IoT Device Tester for Amazon FreeRTOS, to run the test project you do not need to modify this file.

2.3.2 "Hello World" Demo and Getting Started Guide

Prepare the "Hello World" demo project in the code packaged delivered to Amazon. The project creation process is similar to the test project creation. Please see Appendix N: "Hello World" Demo Project Set Up for detailed instructions.

Prepare a "Getting Started Guide" for your board to help users run the Hello World Demo project (and any other demos you may include). You can use the Getting Started Guide template to start and look at the guide for the Window Simulator for reference.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

2.3.3 Configure your board name

Please put your board name in:
$AFR_HOME/demos/[vendor]/[board]/common/config_files/FreeRTOSConfig.h

```
#define mqttconfigMETRIC_PLATFORM    "Platform=Unknown"
```

Replace "Unknown" with your own board name.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

3 FAQs

1. **What is an Amazon FreeRTOS port?**
   An Amazon FreeRTOS port is a board-specific implementation of APIs for certain Amazon FreeRTOS libraries. The port enables these APIs to work on the specific board, and implements the required integration with device drivers and BSPs provided by the platform vendor. It should also include any configuration adjustments (e.g. clock rate, stack size, heap size) required by the board.

2. **Do I need to retest for minor version releases of Amazon FreeRTOS?**
   There is no need to retest for qualification with minor version releases of Amazon FreeRTOS.

3. **What network ports will need to be opened to run AFQ tests?**
   The network connections needed in the AFQ tests include

Port	Protocol
`443, 8883`	MQTT
8443	Greengrass Discovery

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

4 Contact Us

If you have any questions, please post questions to AWS Amazon FreeRTOS forum (you must have an AWS account): http://forums.aws.amazon.com.

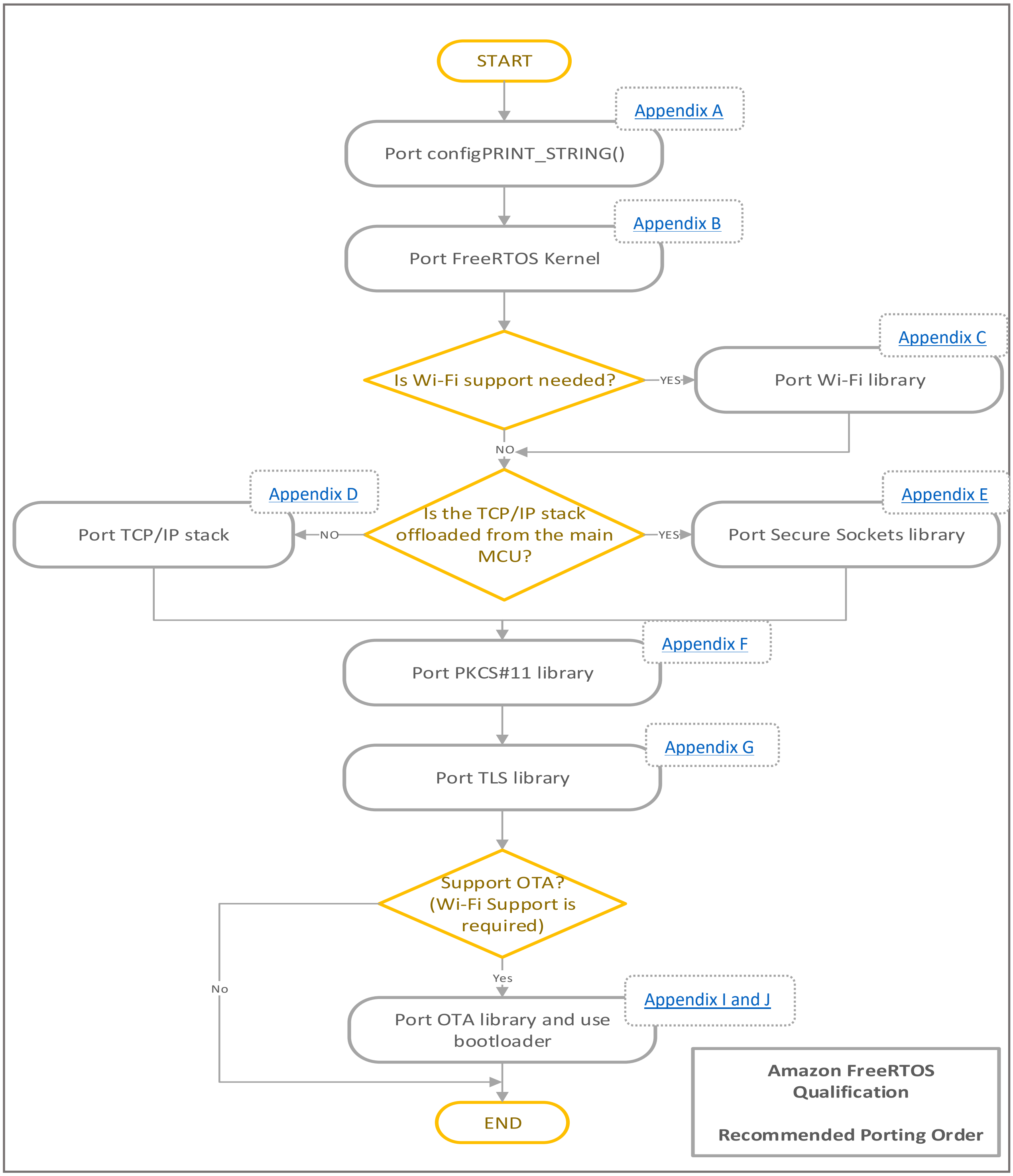
You can also post questions in GitHub/issues board (you must have a GitHub account): https://github.com/aws/amazon-freertos/issues.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5 Appendix

The Appendix contains detailed descriptions of Amazon FreeRTOS libraries and macros to be ported, as well as steps to verify them.

Here is the recommended porting order:


START
Appendix A
Port configPRINT_STRING()
Appendix B
Port FreeRTOS Kernel
Is Wi-Fi support needed?
YES
Appendix C
Port Wi-Fi library
NO
Appendix D
Port TCP/IP stack
NO
Is the TCP/IP stack offloaded from the main MCU?
YES
Appendix E
Port Secure Sockets library
Appendix F
Port PKCS#11 library
Appendix G
Port TLS library
Support OTA? (Wi-Fi Support is required)
No
Yes
Appendix I and J
Port OTA library and use bootloader
END
Amazon FreeRTOS Qualification
Recommended Porting Order

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.1 Appendix A: configPRINT_STRING()

Description

`configPRINT_STRING()` is a macro used by the AFQ test framework to output test results as human readable ASCII strings. It must be implemented before AFQ porting and testing can begin.
These instructions assume test results are output over a UART serial port.

Pre-requisites

1. A development board that supports UART or virtual COM port output.
2. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied UART initialization and output functions.
3. The UART initialization and output must not have any dependency on FreeRTOS.

Setup

1. Connect a terminal emulator, such as TeraTerm, to the port on the target hardware that is to be used to output test results.

Porting

1. Locate the call to `configPRINT_STRING`(“Test Message”) within the function prvMiscInitialization(), which is itself in the file `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c.`
2. Immediately before the call to `configPRINT_STRING`(“Test Message”), add code that uses the vendor-supplied UART driver to initialize the UART to 115200 baud
3. `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` contains an empty definition of `configPRINT_STRING().` The macro takes a NULL terminated ASCII C string as its only parameter. Update the empty definition of `configPRINT_STRING()` so that it calls the vendor-supplied UART output function. For example, if the UART output function has the following prototype:

```
void MyUARTOutput( char *DataToOutput, size_t LengthToOutput );
```

then you would implement `configPRINT_STRING()` as:

```
#define configPRINT_STRING( X ) MyUARTOutput( (X), strlen( (X) ) )
```

Manual Testing

Build and execute the application. If “Test Message” appears in the UART console then the console is connected and configured correctly, and `configPRINT_STRING()` is behaving as expected. If this is the case then `configPRINT_STRING()` testing is complete and the call to `configPRINT_STRING( “Test Message” )` can be removed from `prvMiscInitialization()`.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.2 Appendix B: FreeRTOS kernel

Description

Amazon FreeRTOS uses the FreeRTOS kernel for multitasking and inter-task communications. This appendix describes how to integrate a port of the FreeRTOS kernel into the AFQ test project.

The FreeRTOS.org website contains a list of all the available kernel ports.

Porting the FreeRTOS kernel to a new architecture is out of scope of this document. Contact the Amazon FreeRTOS Qualification team if a port does not exist for your architecture.

Pre-requisites

1. An official FreeRTOS kernel port for the target MCU architecture.
2. A test project that was created in accordance with the instructions provided in the body of this document (Create the Test Project), and that includes the correct FreeRTOS kernel port files for the MCU and compiler in use.
3. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.

Porting

The header file `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` contains application specific FreeRTOS kernel configuration settings. The FreeRTOS.org website provides a description of each configuration option. In particular, ensure the following constants are set correctly for your hardware:

Configuration definitions	**Comment**
`configCPU_CLOCK_HZ`	The frequency of the clock used to generate the tick interrupt.
`configMINIMAL_STACK_SIZE`	As a starting point, this can be set to whichever value is used in the official FreeRTOS demo for the FreeRTOS kernel port in use. Official FreeRTOS demos are those distributed from the FreeRTOS.org web site. Ensure stacks overflow checking is set to 2, and increase configMINIMAL_STACK_SIZE if overflows occur. To save RAM, set stack sizes to the minimum value that does not result in a stack overflow.
`configTOTAL_HEAP_SIZE`	Sets the size of the FreeRTOS heap. Like task stack sizes, the heap size can be tuned to ensure unused heap space does not consume RAM.

ARM Cortex-M3, M4 and M7 devices must also have `configPRIO_BITS and configMAX_SYSCALL_INTERRUPT_PRIORITY` set correctly.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

⚠ The MQTT (Appendix H) and Secure Sockets (Appendix E) libraries have not been ported yet, so it is necessary to comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()` and `SOCKETS_Init()` from within function `SYSTEM_Init()`, which is implemented in `$AFR_HOME/lib/utils/aws_system_init.c`.

Test Setup

No further setup is required for this section.

Test Execution

1. Build and execute the project.
2. If a "." appears in the UART console every 5 seconds then the FreeRTOS kernel is operating as expected and this test is complete. Set configUSE_IDLE_HOOK to 0 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` before moving to the next section. Setting configUSE_IDLE_HOOK to 0 stops the FreeRTOS kernel from executing vApplicationIdleHook(), and so stop the "." Being output during future test executions.
3. If a "." appears at any other frequency then check the setting of configCPU_CLOCK_HZ in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h`. `configCPU_CLOCK_HZ` must be set to the correct value for your board.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.3 Appendix C: Wi-Fi Management

Description

The Wi-Fi Management library is the Amazon FreeRTOS interface to vendor-supplied Wi-Fi drivers. Skip this section if your hardware does not support Wi-Fi.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied Wi-Fi drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel
4. Two wireless Access Points.

Preparing the IDE Project

1. Add the source file `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` into the `[project_top_level]/lib/aws/wifi` virtual folder of the IDE project.
2. Add the source file `$AFR_HOME/tests/common/wifi/aws_test_wifi.c` into the `[project_top_level]/application_code/common_tests/wifi` virtual folder of the IDE project.
3. Enable the Wi-Fi connection code in `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` by deleting the `#if 0` and `#endif` compiler directives in the functions `vApplicationDaemonTaskStartupHook(void)` and `prvWifiConnect(void)`.

Porting

1. `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` contains empty definitions of a set of Wi-Fi management functions. Use the vendor-supplied Wi-Fi driver library to implement at least the subset of functions listed in the table below. `$AFR_HOME/lib/include/aws_wifi.h` provides the information necessary to complete the implementations.

Function	Description
WIFI_On	Turns on Wi-Fi module. Initializes the drivers
WIFI_ConnectAP	Connects to a Wi-Fi Access Point (AP)
WIFI_Disconnect	Disconnects from the currently connected AP
WIFI_Scan	Performs a Wi-Fi network scan
WIFI_GetIP	Retrieves the Wi-Fi interface's IP address
WIFI_GetMAC	Retrieves the Wi-Fi interface's MAC address
WIFI_GetHostIP	Retrieves the host IP address from a hostname using DNS

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Since the MQTT library is not used for running Wi-Fi tests, comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()`in function `SYSTEM_Init()`in file `$AFR_HOME/lib/utils/aws_system_init.c`. Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()`in function `SYSTEM_Init()`, in file `$AFR_HOME/lib/utils/aws_system_init.c.`

Test Setup

If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

1. In `$AFR_HOME/tests/common/include/aws_clientcredential.h`, set the macros shown in table below to be correct for the first Wi-Fi access point.

Macro name	Value
clientcredentialWIFI_SSID	The Wi-Fi SSID as a C string (in quotes)
clientcredentialWIFI_PASSWORD	The Wi-Fi password as a C string
clientcredentialWIFI_SECURITY	Either eWiFiSecurityOpen, eWiFiSecurityWEP, eWiFiSecurityWPA, or eWiFiSecurityWPA2. eWiFiSecurityWPA2 is recommended.

2. In `$AFR_HOME/tests/common/wifi/aws_test_wifi.c`, set the macros shown in table below to be correct for the second Wi-Fi access point.

Macro name	Value
testwifiWIFI_SSID	The Wi-Fi SSID as a C string (in quotes)
testwifiWIFI_PASSWORD	The Wi-Fi password as a C string
testwifiWIFI_SECURITY	Either eWiFiSecurityOpen, eWiFiSecurityWEP, eWiFiSecurityWPA, or eWiFiSecurityWPA2. eWiFiSecurityWPA2 is recommended.

The Wi-Fi management tests listed in the bullet points below have a dependency on the Secure Sockets library, which may not have been ported yet. If the Secure Sockets library has not been ported then all the Wi-Fi management tests other than those listed in the bullets below must pass. After the Secure Sockets library has been ported it is necessary to re-run the Wi-Fi management tests to ensure all the tests (including those in the bullet points below) pass. Additionally, the tests listed in the bullet points attempt to communicate with an echo server. See the Appendix E: Secure Sockets section "Test Setup" for information on starting the echo server.

- WiFiConnectionLoop
- WiFiIsConnected

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

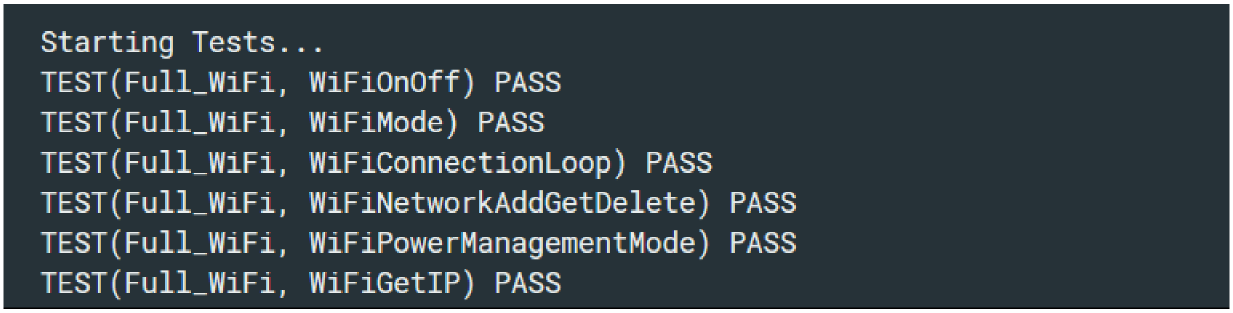
- WiFiConnectMultipleAP
- WiFiSeperateTasksConnectingAndDisconnectingAtOnce

3. Enable the Wi-Fi tests, by setting the `testrunnerFULL_WIFI_ENABLED` macro in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h` to 1.

Test Execution

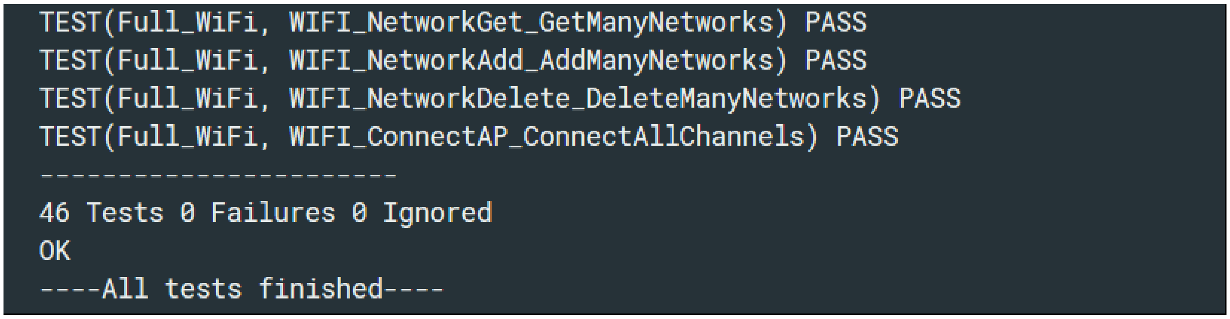
1. Build and execute the test project.
2. View the test results in the UART console. As noted in the 'Test Setup' section of this appendix, not all the tests will pass until porting of the Secure Sockets library is complete. If all the tests that are expected to pass are passing, then save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example output:



```
Starting Tests...
TEST(Full_WiFi, WiFiOnOff) PASS
TEST(Full_WiFi, WiFiMode) PASS
TEST(Full_WiFi, WiFiConnectionLoop) PASS
TEST(Full_WiFi, WiFiNetworkAddGetDelete) PASS
TEST(Full_WiFi, WiFiPowerManagementMode) PASS
TEST(Full_WiFi, WiFiGetIP) PASS
```

…



```
TEST(Full_WiFi, WIFI_NetworkGet_GetManyNetworks) PASS
TEST(Full_WiFi, WIFI_NetworkAdd_AddManyNetworks) PASS
TEST(Full_WiFi, WIFI_NetworkDelete_DeleteManyNetworks) PASS
TEST(Full_WiFi, WIFI_ConnectAP_ConnectAllChannels) PASS
-----------------------
46 Tests 0 Failures 0 Ignored
OK
----All tests finished----
```

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test your implementation of WiFi.

5.4 Appendix D: TCP/IP Stack

Amazon FreeRTOS provides TCP/IP stack support for boards that does not have on-chip TCP/IP functionality. Skip this section if your target hardware offloads TCP/IP functionality to a separate network processor or module.

FreeRTOS+TCP is a native TCP/IP stack working off FreeRTOS Kernel that is actively maintained by AWS FreeRTOS software team. This is recommended TCP/IP stack to use with Amazon FreeRTOS.

The lwIP (lightweight IP) TCP/IP stack is an open source third-party TCP/IP stack. A port layer is added to make it work with FreeRTOS Kernel. The version of lwIP that we currently support is not the latest release.

This appendix only describes how to create a port to the target hardware's Ethernet driver, and test as far as ensuring the Ethernet driver can connect to the network. Actually sending and receiving data is not tested until the Secure Sockets library port is complete.

D.1 FreeRTOS+TCP

Description

FreeRTOS+TCP is the TCP/IP stack used by Amazon FreeRTOS. See https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP for more information.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied Ethernet drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add the source files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source` directory to the [project_top_level]/lib/FreeRTOS-Plus-TCP/source folder of the IDE project.
2. Add the header files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/include` directory to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/include` folder of the IDE project.
3. Add the port source files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/[board_family]/` directory to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/portable/NetworkInterface` folder of the IDE project.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

4. Add the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/BufferManagement/BufferAllocation_2.c` source file to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/portable/BufferManagement` folder of the IDE project.

FreeRTOS has five example heap implementations under `$AFR_HOME /lib/FreeRTOS/portable/MemMang`. Using FreeRTOS+TCP and `BufferAllocation_2.c` requires the heap_4.c implementation.

5. Add the directory `$AFR_HOME/lib/FreeRTOS-Plus-TCP/include` to your compiler's include path.

Porting

1. Check the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/` directory to see if a port to your target hardware already exists.
2. If a port does not exist, then:
   a. Rename the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/[board_family]` directory to be appropriate for the target hardware.
   b. Follow the instruction on the FreeRTOS.org website for porting the TCP/IP stack to a different microcontroller, and if necessary, a different compiler, to create a new port that uses the vendor supplied Ethernet drivers. Implement the new port in a file called NetworkInterface.c, and save the file in the newly renamed directory.

   Note: The files in `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/BufferManagement` are used by multiple ports so must not be edited.

3. Update the FreeRTOS+TCP configuration file `$AFR_HOME/tests/[vendor/[board]/common/config_files/FreeRTOSIPConfig.h` so it is correct for your target hardware. The FreeRTOS.org website describes each configuration option.

⚠ Since the MQTT library is not used for running Wi-Fi tests, comment out the lines that call `BUFFERPOOL_Init(), MQTT_AGENT_Init()`in function `SYSTEM_Init()`in file `$AFR_HOME/lib/utils/aws_system_init.c.` Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()`in function `SYSTEM_Init(),` in file `$AFR_HOME/lib/utils/aws_system_init.c.`

Test Setup

1. In `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c`, uncomment the call to `FreeRTOS_IPInit()` in `main()`. By default, the IP address is acquired by DHCP. If DCHP fails or you do not want to use DHCP, you must set a static IP address in `$AFR_HOME/tests/[vendors]/[board]/common/application_code/main.c`. The following variables must be set to valid values of your actual network:

Network configuration variables	Description
uint8_t ucMACAddress[6]	
uint8_t ucIPAddress[4]	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

uint8_t ucNetMask[4]	
uint8_t ucGatewayAddress[4]	
uint8_t ucDNSServerAddress[4]	
uint8_t ucMACAddress[6]	

2. In `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSIPConfig.h` set the `ipconfigUSE_NETWORK_EVENT_HOOK` macro to 1.
3. In `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` add the code below at the start of `vApplicationIPNetworkEventHook()`:

```
if (eNetworkEvent == eNetworkUp)
{
    configPRINT("Network connection successful.\n\r");
}
```

Test Execution

1. Build and execute the test project.
2. If "Network connection successful" appears in the UART console, then the Ethernet driver has successfully connected to the network and this test is complete.

D.2 lwIP

lwIP is an alternative open source TCP/IP stack. See https://savannah.nongnu.org/projects/lwip/ for more information. The version we currently support is 2.0.3.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied Ethernet drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add the source files from the `$AFR_HOME/lib/third_party/lwip/src` directory to the `[project_top_level]/lib/lwip/` folder of the IDE project.
2. Add the FreeRTOS port source files from the `$AFR_HOME/lib/third_party/lwip/src/portable/arch` directory to the `[project_top_level]/lib/third_party/lwip/src/portable/arch` folder of the IDE project.
3. Add platform specific port files. Add files and directories from `$AFR_HOME/lib/third_party/lwip/src/portable/[vendor]/[board]` directory to the `[project_top_level]lib/third_party/lwip/src/portable/[vendor][board]` directory. If there is a board specific change in a .c file that has the original already added in `[project_top_level]/lib/third_party/lwip/src`, the ported .c file must replace the original .c file in `[project_top_level]/lib/third_party/lwip/src` directory.
4. Add the following directories to your compiler's include path.
    - `$AFR_HOME/lib/third_party/lwip/src/include`
    - `$AFR_HOME/lib/third_party/lwip/src/portable`
    - `$AFR_HOME/lib/third_party/lwip/src/portable/[vendor][board]/include`

Porting

1. Check the `$AFR_HOME/lib/third_party/lwip/src/portable/` directory to see if a port to your target hardware already exists.
2. If a port does not exist, then:
    - Rename the `$AFR_HOME/lib/third_party/lwip/src/portable/[vendor][board]` directory to be appropriate for the target hardware.
    - Copy the network interface stub file $AFR_HOME/lib/third_pary/lwip/src/netif/ethernetif.c to `$AFR_HOME/lib/third_party/lwip/src/portable/[vendor][board]/netif/` and port the file according to the comments in the stub file.
3. The lwIP TCP/IP stack should be initialized calling tcpip_init() after hardware setup.
4. Create the lwIP configuration file `lwipopts.h` in `$AFR_HOME/tests/[vendor/[board]/common/config_files/`. You MUST include "`arch/lwipopts_freertos.h`" where we already defined the options needed for FreeRTOS port of

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

lwIP TCP/IP stack. If you have any platform specific modification, please add into this configuration file.

⚠ Since the MQTT library is not used for running Wi-Fi tests, comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()`in function `SYSTEM_Init()`in file `$AFR_HOME/lib/utils/aws_system_init.c`. Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()`in function `SYSTEM_Init()`, in file `$AFR_HOME/lib/utils/aws_system_init.c`.

Test Setup

N/A

Test Execution

There are no AFQP tests for lwIP module.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.5 Appendix E: Secure Sockets

Description

The Secure Sockets library API is based on the Berkeley sockets API. It provides the API functions necessary to create and configure a TCP socket, connect to an MQTT broker, and send and receive TCP data.

The library is called Secure Sockets as it also encapsulates TLS functionality. To create a TLS protected socket the application writer need only creates a standard TCP socket, then uses a setsockopt call make the socket use TLS.

If your target hardware does not offload TCP/IP functionality to a separate network chip then use the FreeRTOS+TCP TCP/IP stack. A Secure Sockets implementation already exists for the FreeRTOS+TCP TCP/IP stack used in conjunction with mbedTLS – so if you are using those libraries no porting is necessary, but the Secure Sockets tests must still be executed and pass.

Pre-requisites

1. If you are using Wi-Fi for network connectivity: A port of the Wi-Fi management library as described in Appendix C: Wi-Fi Management.
2. A port of the TCP/IP stack library as described in Appendix D: TCP/IP Stack.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. If you use the FreeRTOS+TCP TCP/IP stack, add `$AFR_HOME/lib/secure_sockets/portable/freertos_plus_tcp/aws_secure_sockets.c` to the `[project_top_level]/lib/aws/secure_sockets` folder of the IDE project.
2. If you use the lwIP as your TCP/IP stack, add `$AFR_HOME/lib/secure_sockets/portable/lwip/aws_secure_sockets.c` to the `[project_top_leve]/lib/aws/secure_sockets` folder of the IDE project.
3. If you use your own port, add `$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c` to the `[project_top_leve]/lib/aws/secure_sockets` folder of the IDE project.
4. Add `$AFR_HOME/tests/common/secure_sockets/portable/[vendor]/[board]/aws_test_tcp_portable.h` and `$AFR_HOME/tests/common/secure_sockets/portable/[vendor]/[board]/aws_test_tcp.c` to the `[project_top_level]/application_code/common_tests/secure_sockets` folder of the IDE project.

Porting

If you use the FreeRTOS+TCP TCP/IP stack or lwIP TCP/IP stack then no porting is necessary.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

If your target hardware offloads TCP/IP functionality to a separate network chip then it is necessary to implement all the functions for which stubs already exist in `$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c`. `$AFR_HOME/lib/include/aws_secure_sockets.h` contains the information necessary to create the implementations.

⚠ Since the MQTT library is not used for running Wi-Fi tests, comment out the calls to `BUFFERPOOL_Init()` and `MQTT_AGENT_Init()` from `SYSTEM_Init()`, which is located in `$AFR_HOME/lib/utils/aws_system_init.c.` Make sure the call to `SOCKETS_init()` is uncommented.

Test Setup

If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

1. The Secure Sockets tests require an echo server to be present on the network. For your convenience, the AFQ tests distribution contains a suitable echo server, written in Go, in the `$AFR_HOME/tools/echo_server directory.`

   To start the echo server:

   a) Ensure the latest version of Go is installed on the computer that will run the echo server (the server host). Go can be installed from https://golang.org/dl/
   b) Copy `$AFR_HOME/tools/echo_server/echo_server.go` onto the server host.
   c) Start the server by typing: "`go run echo_server.go`"
   d) In `$AFR_HOME/tests/common/include/aws_test_tcp.h,` set the parameters shown in the table below to the IP address of the server host. The value 192.168.0.200 is an example only.

Echo Server IP address	Example value if address is 192.168.0.200
`tcptestECHO_SERVER_ADDR0`	`192`
`tcptestECHO_SERVER_ADDR1`	`168`
`tcptestECHO_SERVER_ADDR2`	`0`
`tcptestECHO_SERVER_ADDR3`	`200`

To test that the echo server is working, open a command prompt on a computer on the same network and type "telnet 192.168.0.200 9001" from a Windows host, or "nc 192.168.0.200 9001" from a Linux host. Again, use the correct IP address for the server host – 192.168.0.200 is used as an example only.

It may be necessary to adjust the firewall settings on the server host to enable the MCU to connect.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

2. Set the tcptestTCP_ECHO_CLIENT_PORT macro in `$AFR_HOME/tests/common/include/aws_test_tcp.h` to the port on which the echo server is listening. The provided echo server listens on port 9001. If you have problems connecting to port 9001 due to corporate network security policies, you can change the listening port in the echo server code to a port that is not restricted by your company's security policy.
3. Set the tcptestSECURE_SERVER macro to 0 in `$AFR_HOME/tests/common/include/aws_test_tcp.h` to run the socket tests without TLS.
4. Set the testrunnerFULL_TCP_ENABLED macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner.config.h` to enable the sockets tests.
5. Enable the testing task in `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` by deleting the `#if 0` and `#endif` compiler directives, in `vApplicationIPNetworkEventHook ( void )`. This change is required for all the remaining libraries to be ported in this document.

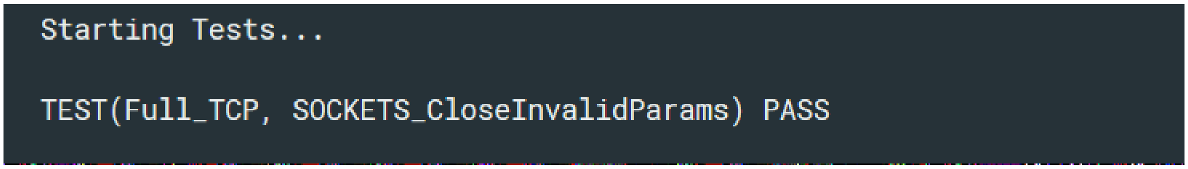
Dependency on TLS

The test set for this library (Secure Sockets) includes some tests that require TLS which is described later in this document. The functionality verified by these tests is exercised when tcptestSECURE_SERVER macro is set to 1. You MUST come back to run this subset of the Secure Socket tests after TLS library porting is completed. A TLS capable echo server should be setup before running these tests (Appendix M: TLS Server Setup).

Test Execution

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example test results output:

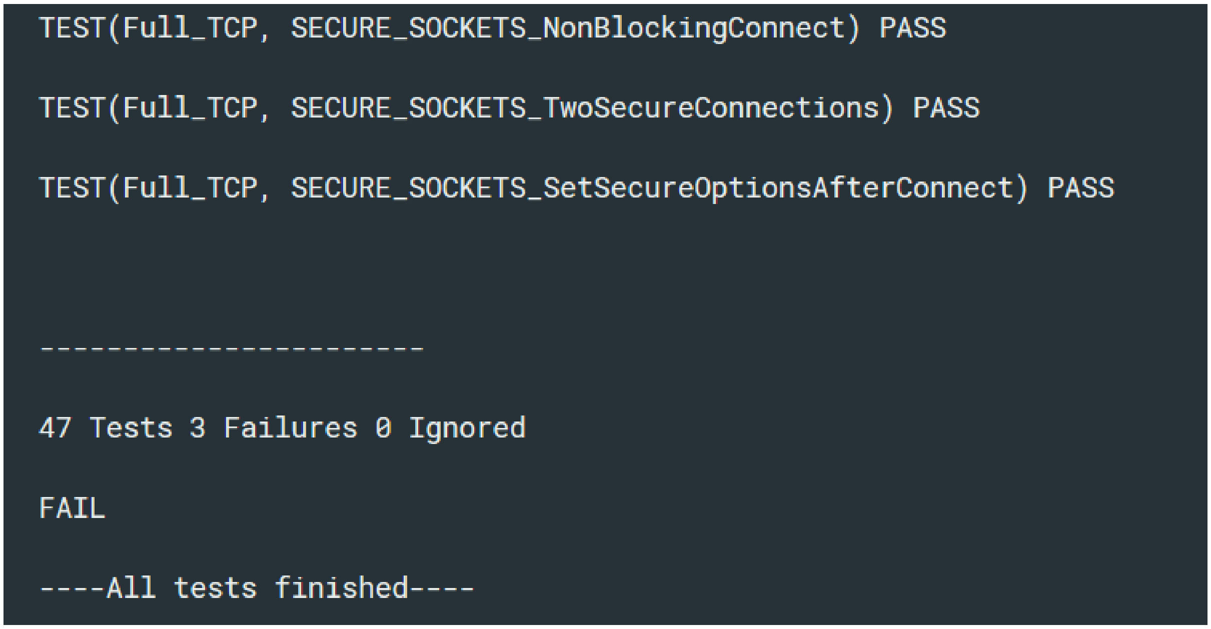


```
Starting Tests...

TEST(Full_TCP, SOCKETS_CloseInvalidParams) PASS
```

…

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test your implementation of Secure Sockets.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.



```
TEST(Full_TCP, SECURE_SOCKETS_NonBlockingConnect) PASS

TEST(Full_TCP, SECURE_SOCKETS_TwoSecureConnections) PASS

TEST(Full_TCP, SECURE_SOCKETS_SetSecureOptionsAfterConnect) PASS



-----------------------

47 Tests 3 Failures 0 Ignored

FAIL

----All tests finished----
```

5.6 Appendix F: PKCS #11

Description

Amazon FreeRTOS uses the open standard PKCS #11 "CryptoKi" API as the abstraction layer for cryptographic operations, including:

- Encryption and decryption.
- Storage and enumeration of X.509 certificates.
- Storage and management of cryptographic keys.

See the open standard PKCS #11 Cryptographic Token Interface Base Specification:

http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html

Storing private keys in general purpose flash memory can be convenient in evaluation and rapid prototyping scenarios. However, when it comes to production scenarios, we recommend the use of dedicated cryptographic hardware in order to reduce the threats of data theft and device duplication. Cryptographic hardware includes components with features that prevent cryptographic secret keys from being exported. In order to use such hardware with Amazon FreeRTOS, the PKCS #11 API must be ported to it.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied storage drivers that are suitable for sensitive data.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add `$AFR_HOME/lib/pkcs11/portable/[vendor]/[board]/aws_pkcs11_pal.c` to the `[project_top_level]/lib/aws/pkcs11` folder of the test project.
2. Add the PKCS #11 library header files from `$AFR_HOME/lib/third_party/pkcs11` to the `[project_top_level]/lib/third_party/pkcs11` folder of the test project.
3. Add the PKCS #11 tests from `$AFR_HOME/tests/common/pkcs11/aws_test_pkcs11.c` to the `[project_top_level]/application_code/common_tests/pkcs11` folder folder of the test project.
4. Add the implementation of PKCS #11 for mbedTLS `$AFR_HOME/lib/pkcs11/mbedtls/aws_pkcs11_mbedtls.c` to the `[project_top_level]/lib/pkcs11` folder of the test project
5. Import the CRYPTO abstraction wrapper file for mbedTLS `$AFR_HOME/lib/crypto/aws_crypto.c` to the `[project_top_level]/lib/crypto` folder of the test project.
6. Add the mbedTLS library itself from `$AFR_HOME/lib/third_party/mbedtls/library` into the `[project_top_level]/lib/third_party/mbedtls/source` folder of the test project.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

7. Add the mbedTLS library header file from `$AFR_HOME/lib/third_party/mbedtls/include` into the `[project_top_level]/lib/third_party/mbedtls/include` folder of the test project.
8. Add both `$AFR_HOME/lib/third_party/mbedtls/include` and `$AFR_HOME/lib/third_party/pkcs11` to the compiler's include path.

Porting

1. Porting the PKCS #11 API functions

   The PKCS #11 API is dependent on the implementation of cryptographic primitives such as SHA256 hashing and ECDSA signing. The Amazon FreeRTOS implementation of PKCS #11 uses the cryptographic primitives implemented in the mbedTLS library, for which a port is already provided. Modifying the existing PKCS #11 port is required if you wish to use a different software implementation of the cryptographic primitives (i.e., other than mbedTLS), or if your target hardware offloads crypto to a separate module.

2. Porting the PKCS #11 Platform Abstraction Layer (PAL) for device specific certificate and key storage

   If you decide to use the Amazon FreeRTOS implementation of PKCS #11, there is a relatively small amount of customization required in order to read and write cryptographic objects to non-volatile memory (NVM) (for example, onboard flash memory).

   Cryptographic objects should be stored in a section of NVM that is not initialized/erased on device reprogramming. Users of the PKCS #11 library should be able to provision devices with credentials, and then reprogram the device with a new application which accesses these credentials via the PKCS #11 interface.

   PKCS #11 PAL ports must provide a location to store device client certificate, device client private key, device client public key, trusted root CA, and a code verification public key (or certificate containing the code verification public key) for secure bootloader and over-the-air updates.

   `$AFR_HOME/lib/pkcs11/portable/[vendor][board]/aws_pkcs11_pal.c` contains stubs for the PAL functions, of which you must provide ports for at least the functions listed in the table below:

Function	**Description**
PKCS11_PAL_SaveObject	Writes data to non-volatile storage
PKCS11_PAL_FindObject	Uses a PKCS #11 CKA_LABEL to search for a corresponding PKCS #11 object in non-volatile storage, and returns that object's handle if it exists.
PKCS11_PAL_GetObjectValue	Retrieves the value of an object, given the handle.
PKCS11_PAL_GetObjectValueCleanup	Cleanup for the PKCS11_PAL_GetObjectValue call. May be used to free memory allocated in PKCS11_PAL_GetObjectValue call.

3. Implement `mbedtls_hardware_poll()`

   ⚠ You only need to port this function if you plan to use Amazon FreeRTOS' PKCS#11 implementation and the mbedTLS library for underlying cryptographic and TLS support.

   TCP/IP and TLS require cryptographic pseudo-random number generation (PRNG) for sequence number and key generation, respectively. A hardware entropy source is important for seeding the PRNG. For the mbedTLS library to work, you MUST implement mbedtls_hardware_poll() which allows the mbedTLS library to seed its PRNG using your board's entropy source. This function is located in `$AFR_HOME/lib/pkcs11/portable/[vendor][board]/aws_pkcs11_pal.c`

   Note that if the port board does not contain a cryptographically accepted random number source, it may be necessary to follow the entropy seeding approach described in the mbedTLS porting guide below.

   For more information see, https://docs.mbed.com/docs/mbed-os-handbook/en/5.2/advanced/tls_porting.

⚠ Since the MQTT library is not used for running PKCS11 tests, comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()`in function `SYSTEM_Init()`in file `$AFR_HOME/lib/utils/aws_system_init.c.` Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()`in function `SYSTEM_Init()`, in file `$AFR_HOME/lib/utils/aws_system_init.c.`

Test Setup

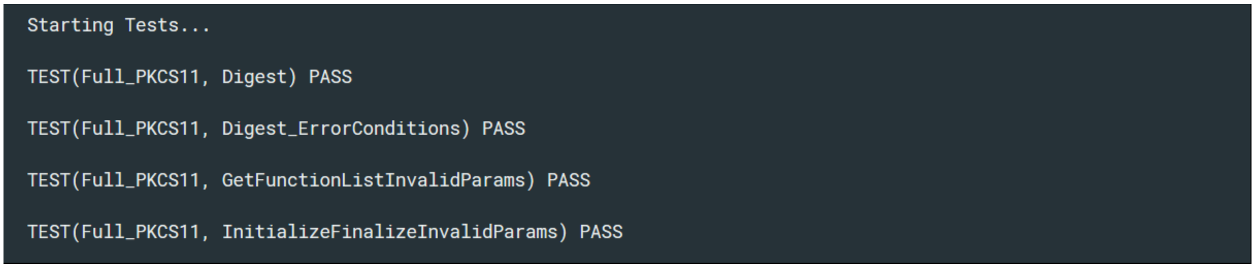
If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

1. Enable the PKCS 11 test by setting the `testrunnerFULL_PKCS11_ENABLED` macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`.

Test Execution

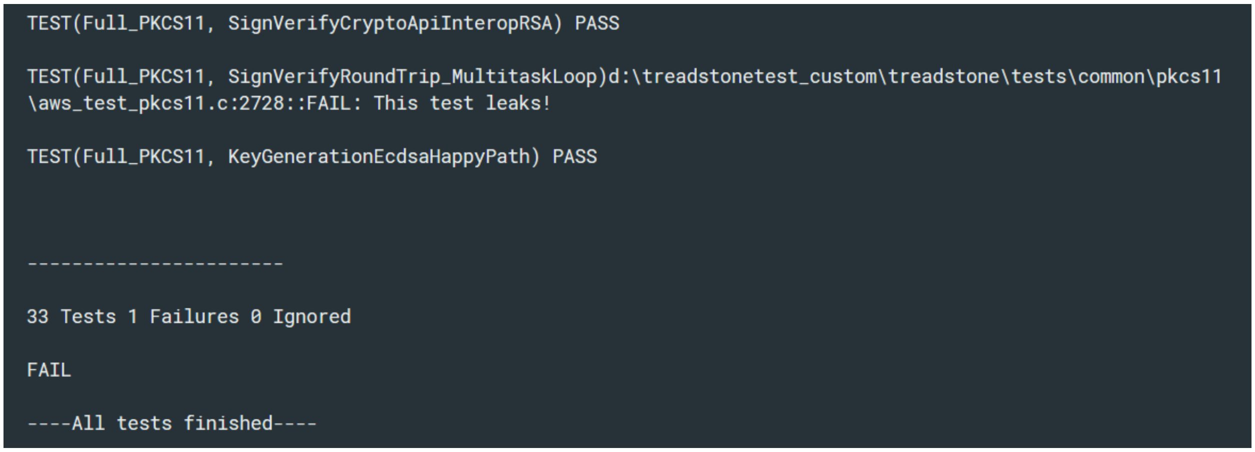
Build and execute the project. The UART output indicates how many tests have run and completed successfully. Copy the results from the terminal and save it to a text file.

Example of the test results output:



```
Starting Tests...

TEST(Full_PKCS11, Digest) PASS

TEST(Full_PKCS11, Digest_ErrorConditions) PASS

TEST(Full_PKCS11, GetFunctionListInvalidParams) PASS

TEST(Full_PKCS11, InitializeFinalizeInvalidParams) PASS
```

…



```
TEST(Full_PKCS11, SignVerifyCryptoApiInteropRSA) PASS

TEST(Full_PKCS11, SignVerifyRoundTrip_MultitaskLoop)d:\treadstonetest_custom\treadstone\tests\common\pkcs11
\aws_test_pkcs11.c:2728::FAIL: This test leaks!

TEST(Full_PKCS11, KeyGenerationEcdsaHappyPath) PASS



-----------------------

33 Tests 1 Failures 0 Ignored

FAIL

----All tests finished----
```

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test your implementation of PKCS#11.

5.7 Appendix G: TLS

Description

The AWS IoT Core MQTT broker only accepts mutually authenticated TLS connections. Amazon FreeRTOS can use either mbedTLS, in which case no porting is necessary, or an off-chip TLS implementation, such as those found on some network co-processors. To allow both options the TLS library is not accessed directly, but through a TLS abstraction layer.

In all cases, the TLS tests must be executed and pass. Preparing the tests requires IoT device configuration in the AWS cloud and certificate and key provisioning on the target hardware.

Pre-requisites

1. A port of the Secure Sockets library, as described in Appendix E: Secure Sockets.
2. A port of the PKCS #11 library, as described in Appendix F: PKCS #11.
3. An AWS account.

Preparing the IDE Project

1. Add the TLS abstraction implementation `$AFR_HOME/lib/tls/aws_tls.c` or `$AFR_HOME/lib/tls/portable/[vendor]/[board]/aws_tls.c` (if your target hardware offloads TLS to a separate processor) to the `[project_top_leve]/lib/aws/tls` folder of the test project.
2. Add the TLS tests file `$AFR_HOME/tests/common/tls/aws_test_tls.c` to the `[project_top_level]/application_code/common_tests/tls` folder of the test project.
3. Enable the tests by setting the testrunnerFULL_TLS_ENABLED macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h.`

Porting

If your target hardware offloads TLS functionality to a separate network chip then it is necessary to implement all the TLS abstraction layer functions in the table below. `$AFR_HOME/lib/include/aws_tls.h` contains the information necessary to create the implementations. Save the created file as `$AFR_HOME/lib/tls/portable/[vendor]/[board]/aws_tls.c`

Function	**Description**
TLS_Init	Initialize the TLS context
TLS_Connect	Negotiate TLS and connect to the server
TLS_Recv	Read the requested number of bytes from the TLS connection
TLS_Send	Write the requested number of bytes to the TLS connection
TLS_Cleanup	Free resources consumed by the TLS context

⚠ Since the MQTT library is not used for running Wi-Fi tests, comment out the calls to `BUFFERPOOL_Init() and MQTT_AGENT_Init()` from `SYSTEM_Init(), which is located`

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

in `$AFR_HOME/lib/utils/aws_system_init.c`. Make sure the call to `SOCKETS_init()` is uncommented.

Test Setup

If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

The tests in this section require use of the online AWS console, where your target hardware will be represented as a 'thing', and communicate with AWS via a custom MQTT endpoint that is tied to your AWS account.

The steps below create the certificates and keys necessary to complete qualification tests.

⚠ The tests require the created certificates and keys to be built into the target hardware's executable image. That is convenient in this test scenario, but is **not recommended for production scenarios**, where the keys should be kept in hardened storage.

Certificate Formatting Tool: It is necessary to convert the certificates and keys to C strings before building them into the executable image. The AFQ tests include a tool for that purpose. To convert certificate and key pairs into C strings:

a. Open `$AFR_HOME/tools/certificate_configuration/PEMfileToCString.html` in a web browser.
b. Follow the instructions on the opened web page to load the certificate and private key.
c. Once loaded, follow the instruction in the opened web page to convert the opened certificate and private key to a formatted C string.

1. Set the `clientcredentialMQTT_BROKER_ENDPOINT`[] variable in `$AFR_HOME/tests/common/include/aws_clientcredential.h` to the custom end point of your AWS account – this is the URL the TLS tests connect to.

   To find your custom end point, use the URL `https://aws.amazon.com/iot/` to log into your AWS account, then click the "Settings" link in the bottom left corner of the screen to open the settings window – the customer end point is displayed at the top of the settings window.

2. Noting the information below about the information you need to record during the process, follow the steps in the AWS IoT Getting Started tutorial to create the resources in AWS IoT that will represent your target hardware (Thing, Certificate and Policies).
   a. Start here: `https://docs.aws.amazon.com/iot/latest/developerguide/iot-console-signin.html`, and continue through each of the steps of the tutorial until you complete "Attach your Certificate to a Thing". See notes below for additional guidance about these steps.
   b. During this process,
      - Set the `clientcredentialIOT_THING_NAME` variable in `$AFR_HOME/tests/common/include/aws_clientcredential.h` to the name you assigned your 'thing' (the thing name).

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

- The steps on the link above include the creation of a certificate. Download and save all three files that are generated during that process.
- The steps on the link above include creating a policy. Use the following policy to attach to the certificate:

```
{
    "Effect": "Allow",

    "Action": "iot:*",

    "Resource": "*"

}
```

⚠ This policy will allow all IoT actions on all resources. That is convenient in a test and evaluation scenario but is not recommended for production scenarios.

3. Prepare certificate/key pairs for various tests for the TLS library.

AWS IoT can use AWS IoT-generated certificates or certificates signed by a CA certificate for device authentication. In order to run the various tests, you will need to create multiple credentials as listed below:

a. Generate a certificate by AWS IoT. (RSA Certificate)
b. Generate a certificate from a CSR (Certificate Signing Request). (ECDSA cert, Malformed cert)
c. Generate a certificate from a registered CA. (Untrusted, BYOC certificate)

The credentials for the types listed above are described in detail in following table. All of the testing client certificates and private keys listed above must be stored in:

```
$AFR_HOME/tests/common/aws_clientcredential_keys.h

$AFR_HOME/tests/common/aws_test_tls.h
```

Cert/Key Variables	Description
keyCLIENT_CERTIFICATE_PEM	Device certificate used for AWS cloud communication. It is also used in TLS_ConnectRSA() test. You can set this up using AWS Console.
keyCLIENT_PRIVATE_KEY_PEM	Device private key used for AWS cloud communication. It is also used in TLS_ConnectRSA() test. You can set this up using AWS Console.
tlstestCLIENT_CERTIFICATE_PEM_EC	Certificate for P-256 elliptic curve key. It is used in TLS_ConnectEC() test.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

tlstestCLIENT_PRIVATE_KEY_PEM_EC	A p-256 elliptic curve key. It is used in TLS_ConnectEC() test.
tlstestCLIENT_CERTIFICATE_PEM_MALFORMED	A RSA or ECDSA certificate that has a field modified. Used in TLS_ConnectMalformedCert() test.
tlstestCLIENT_UNTRUSTED_CERTIFICATE_PEM	A certificate is not trusted (not registered) by AWS IoT. Used in TLS_ConnectUntrustedCert().
tlstestCLIENT_UNTRUSTED_PRIVATE_KEY_PEM	The private key correspond to the untrusted certificate.
tlstestCLIENT_BYOC_CERTIFICATE_PEM	A certificate created by a CA (registered to AWS IoT). Used in TLS_ConnectBYOCCredentials().
tlstestCLIENT_BYOC_PRIVATE_KEY_PEM	The private key corresponding to the BYOC certificate.

Setup for RSA certificate/private key used in TLS_ConnectRSA() :

This pair of certificate/key is generated in Test Setup Step 2. The three files you downloaded during that "Thing" creation process will be used here.

Format the certificate and the private key with the formatting tool and copy them to macros

keyCLIENT_CERTIFICATE_PEM

keyCLIENT_PRIVATE_KEY_PEM

in file `$AFR_HOME/tests/common/aws_clientcredential_keys.h.`

Setup for ECDSA certificate/private key used in TLS_ConnectEC():

OpenSSL is an open source toolkit for TLS protocol. (https://www.openssl.org/). We will use openssl in examples of generating certificates below. Please use TLS V1.2. You can download it here: https://www.openssl.org/source/

1) Create a CSR with openssl:
   a) `openssl ecparam -name prime256v1 -genkey -noout -out p256_privatekey.pem`
   b) `openssl req –new –key p256_privatekey.pem –out csr.csr`
2) Create a certificate with the AWS IoT console:
   a) On the AWS IoT / Security / Certificate page, click "Create" in upper right-hand corner
   b) Click "Create with CSR" and upload the .csr file created in step 1.
   c) Download the cert .pem file, activate it and attach the same policy you used when setting up the RSA certificate.
   d) Attach the certificate to the IoT thing created when you set up the "Thing".
3) Format the certificate and the private key using the formatting tool.
4) Copy the created cert and private key to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_CERTIFICATE_PEM_EC`
   b) `tlstestCLIENT_PRIVATE_KEY_PEM_EC`

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5) In `$AFR_HOME/tests/common/tls/aws_test_tls.c`, set the `tlstestMQTT_BROKER_ENDPOINT_EC` variable to the same AWS IoT message broker endpoint address in [TLS Test Setup Step 1](#).

Setup for Malformed certificate used in TLS_ConnectMalformedCert():

The purpose of the test is to be able to use a malformed certificate to authenticate with the server. Random modification of a certificate will most likely be rejected by x509 certificate verification before the connection request is sent out. We have a suggestion to setup this malformed certificate: modifying the issuer of the certificate.

See Appendix S: Modify issuer in a certificate for process details.

Setup for BYOC (Bring You Own Certificate) certificate used in TLS_ConnectBYOCCredentials():

1) Create your own certificate with a valid rootCA/CA chain. See example in Appendix Q: Instructions to Create a BYOC (ECDSA).
2) Register CAs and your own certificate in the AWS IoT console: IoT Core / Secure / Certificates / Create / Get started.
3) Format the certificate and the private key using the [formatting tool](#).
4) Copy the certificate and private key strings to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_BYOC_CERTIFICATE_PEM`
   b) `tlstestCLIENT_BYOC_PRIVATE_KEY_PEM`

Setup for Untrusted certificate used in TLS_ConnectUntrustedCert():

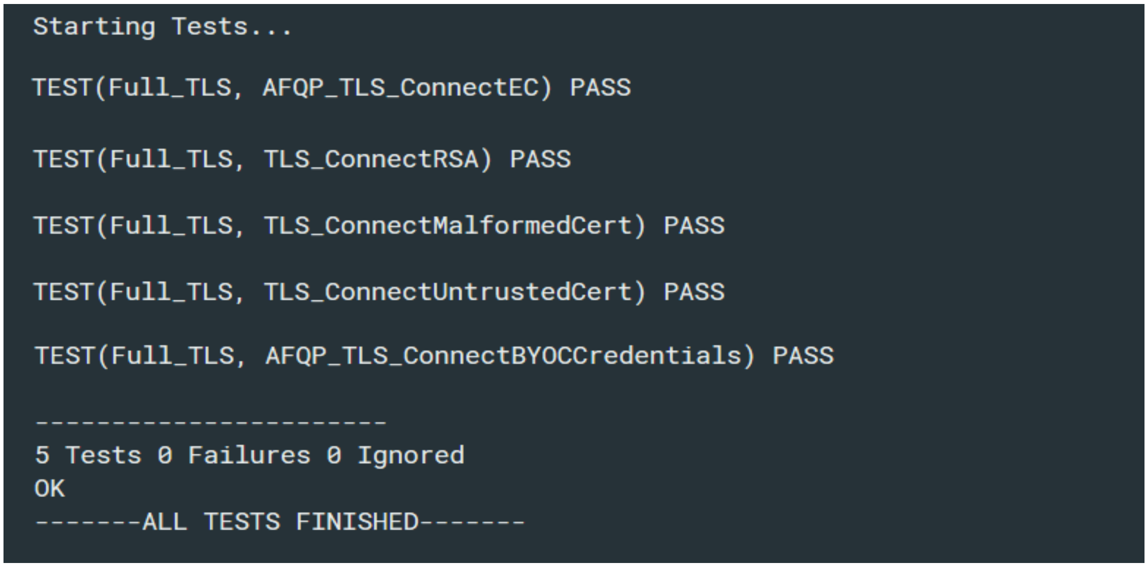
1) Create your own certificate with valid rootCA/CA chain. See example in Appendix Q: Instructions to Create a BYOC (ECDSA).
2) Do not register them in AWS IoT console.
3) Format the certificate and the private key using the [formatting tool](#).
4) Copy the cert and private key strings to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_UNTRUSTED_CERTIFICATE_PEM`
   b) `tlstestCLIENT_UNTRUSTED_PRIVATE_KEY_PEM`

Test Execution

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example of the test results output:

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.



```
Starting Tests...

TEST(Full_TLS, AFQP_TLS_ConnectEC) PASS

TEST(Full_TLS, TLS_ConnectRSA) PASS

TEST(Full_TLS, TLS_ConnectMalformedCert) PASS

TEST(Full_TLS, TLS_ConnectUntrustedCert) PASS

TEST(Full_TLS, AFQP_TLS_ConnectBYOCCredentials) PASS

-----------------------
5 Tests 0 Failures 0 Ignored
OK
-------ALL TESTS FINISHED-------
```

⚠ Once TLS porting and verification is completed, note that you must go back to run a subset of the Secure Socket tests which depend on this functionality i.e. when tcptestSECURE_SERVER macro is set to 1. See Dependency on TLS in the Secure Sockets porting section.

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test your implementation of TLS.

5.8 Appendix H: MQTT

Description

Communication between IoT devices and AWS IoT Core (the MQTT broker) uses the MQTT protocol. The MQTT library that implements the protocol does not need to be ported, but does need to pass all the MQTT tests.

The MQTT library has a dependency on the Buffer Pool library, which is used to allocate the memory necessary to hold MQTT packets.

Pre-requisites

1. A port of the Secure Sockets library, as described in Appendix E: Secure Sockets.
2. A port of the PKCS #11 library, as described in Appendix F: PKCS #11.
3. A port of the TLS library, as described in Appendix G: TLS.
4. An AWS account.
5. An IoT thing created in AWS cloud and its associated credential information. (Refer to TLS_Test_Setup_Step_1, TLS_Test_Setup_Step_2 and RSA_Certificate_Setup)

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add the MQTT library source files from `$AFR_HOME/lib/mqtt` into the `[project_top_level]/lib/aws/mqtt` folder of the test project.
2. Add the Bufferpool source files from `$AFR_HOME/lib/bufferpool` into the `[project_top_level]/lib/bufferpool` folder of the test project.
3. Add the MQTT test source files from `$AFR_HOME/tests/common/mqtt/` to `[project_top_level]/application_code/common_tests/mqtt` folder of the test project.
4. Uncomment all the initialization functions called from `SYSTEM_Init() within $AFR_HOME/lib/utils/aws_system_init.c`.

Porting

⚠ In order to enable MQTT functionality, uncomment the calls to `BUFFERPOOL_Init()` and `MQTT_AGENT_Init()` from `SYSTEM_Init()`, which is located in `$AFR_HOME/lib/utils/aws_system_init.c`. Make sure the call to `SOCKETS_init()` is also still uncommented.

There is no additional porting required for this library.

Test Setup

If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

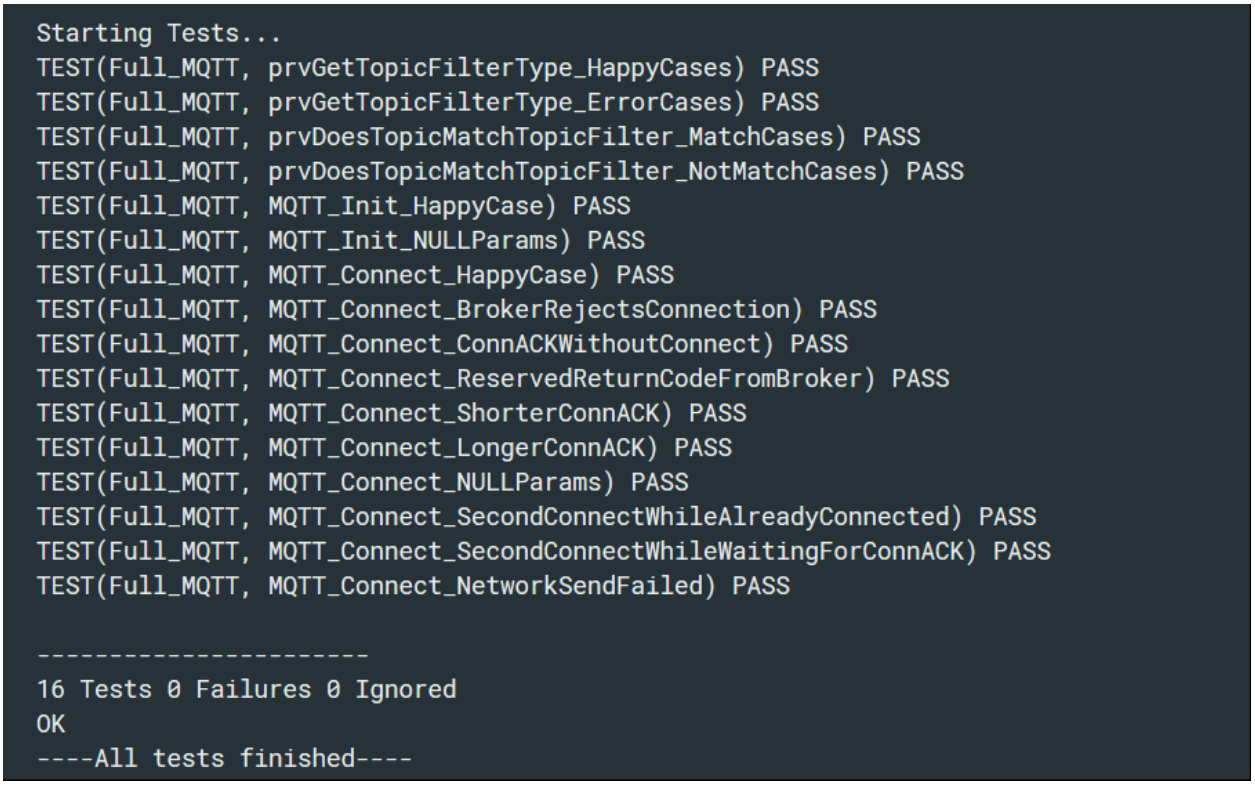
These tests require the certificates and keys that were created prior to testing the TLS library.

1. Enable the MQTT tests by setting testrunnerFULL_MQTT_ENABLED to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`

Test Execution

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file.

Example of the test results output:



```
Starting Tests...
TEST(Full_MQTT, prvGetTopicFilterType_HappyCases) PASS
TEST(Full_MQTT, prvGetTopicFilterType_ErrorCases) PASS
TEST(Full_MQTT, prvDoesTopicMatchTopicFilter_MatchCases) PASS
TEST(Full_MQTT, prvDoesTopicMatchTopicFilter_NotMatchCases) PASS
TEST(Full_MQTT, MQTT_Init_HappyCase) PASS
TEST(Full_MQTT, MQTT_Init_NULLParams) PASS
TEST(Full_MQTT, MQTT_Connect_HappyCase) PASS
TEST(Full_MQTT, MQTT_Connect_BrokerRejectsConnection) PASS
TEST(Full_MQTT, MQTT_Connect_ConnACKWithoutConnect) PASS
TEST(Full_MQTT, MQTT_Connect_ReservedReturnCodeFromBroker) PASS
TEST(Full_MQTT, MQTT_Connect_ShorterConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_LongerConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_NULLParams) PASS
TEST(Full_MQTT, MQTT_Connect_SecondConnectWhileAlreadyConnected) PASS
TEST(Full_MQTT, MQTT_Connect_SecondConnectWhileWaitingForConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_NetworkSendFailed) PASS

-----------------------
16 Tests 0 Failures 0 Ignored
OK
----All tests finished----
```

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test MQTT.

5.9 Appendix I: OTA Updates

Description

The Amazon FreeRTOS over-the-air (OTA) update feature enables you to:

- Deploy new firmware images to a single device, a group of devices, or your entire fleet.
- Deploy firmware to devices as they are added to groups, are reset, or are re-provisioned.
- Verify the authenticity and integrity of new firmware after it has been deployed to devices.
- Monitor the progress of a deployment.
- Debug a failed deployment.
- Digitally sign firmware using the AWS Signer service.

Amazon FreeRTOS devices must enforce cryptographic code-sign verification of the OTA firmware images they receive. Regarding algorithm selection, we recommend the use of the Elliptic-Curve Digital Signature Algorithm (ECDSA), the NIST P256 curve, and a SHA-256 hash

Pre-requisites

1. A Bootloader that can support OTA update as described in Appendix J: Bootloader.
2. A port of the PKCS #11 library, as described in Appendix F: PKCS #11.
3. A port of the TLS library, as described in Appendix G: TLS.

Preparing the IDE Project

1. Add the OTA library files from $AFR_HOME/lib/ota into the IDE project under the `[project_top_level]/lib/aws/ota` virtual folder.
2. Import the OTA library PAL files, `$AFR_HOME/lib/ota/portable/[vendor]/[board]/aws_ota_pal.c` into the IDE project under the `[project_top_level]/lib/aws/ota` virtual folder.

3. Import the OTA tests,
   - `$AFR_HOME/tests/common/ota/aws_test_cbor.c`
   - `$AFR_HOME/tests/common/ota/aws_test_ota_agent.c`
   - `$AFR_HOME/tests/common/aws_test_pal.c`

   into the IDE project under the `[project_top_level]/application_code/common_tests/ota` virtual folder.

4. Add the OTA Update demo from $AFR_HOME/demos/common/ota/aws_ota_update_demo.c into the IDE project under the `[project_top_level]/application_code/common_tests/ota` virtual folder.

Porting

Amazon FreeRTOS defines an OTA platform abstraction layer (PAL) in order to ensure that the OTA library can be used on a wide variety of hardware. The OTA PAL interface is listed below.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Function	Description
prvAbort	Aborts an OTA update.
prvCreateFileForRx	Creates a new file to store the data chunks as they are received.
prvCloseFile	Closes the specified file. This may authenticate the file if storage that implements cryptographic protection is being used.
prvWriteBlock	Writes a block of data to the specified file at the given offset. Returns the number of bytes written on success or negative error code.
prvActivateNewImage	Activates or launches the new firmware image. For some ports, if the device is programmatically reset synchronously, this function may not return.
prvSetImageState	Does what is required by the platform to accept or reject the most recent OTA firmware image (or bundle). Refer to your respective board (platform) details and architecture to determine how to implement this function.
prvGetImageState	Gets the state of the OTA update image.

The following two functions are optional if a device has built-in support for them. If not then they need to be implemented in the PAL.

Function	Description
prvCheckFileSignature	Verifies the signature of the specified file.
prvReadAndAssumeCertificate	Reads the specified signer certificate from the file system and returns it to the caller.

Test Setup

If you use AWS IoT Device Tester for Amazon FreeRTOS to run tests you don't need to perform the following setup steps. See https://aws.amazon.com/freertos/device-tester for more information.

The OTA AFQ tests are split into the following two categories: Agent and PAL module tests, and end-to-end functional tests.

1. Agent/PAL tests
   a. Enable the OTA Agent/PAL tests by setting the following macros to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`:
      - `testrunnerFULL_OTA_AGENT_ENABLED`
      - `testrunnerFULL_OTA_PAL_ENABLED`

   b. Select a signing certificate that is appropriate for your device from `$AFR_HOME/tests/common/ota/test_files.`
   Each board has its specific way to provision a device. The certificate will be used for verification in OTA tests.
   Three types of signing certificates are available in the test code. These include RSA/SHA1, RSA/SHA256 and ECDSA/SHA256. Out of the three, the use of ECDSA/SHA256 is recommended for OTA updates. The other two are available for existing platforms only. If you have a different

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

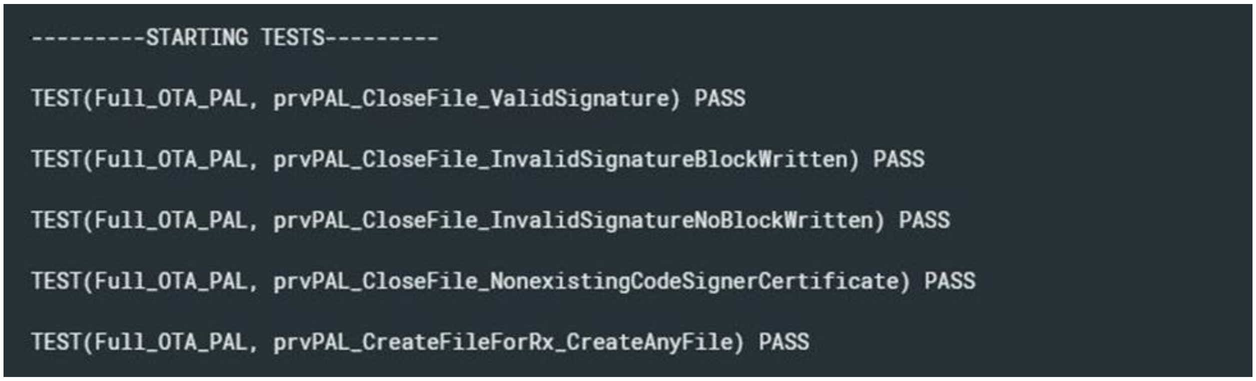
scheme that is not included in the aforementioned three schemes, then please contact your Qual-Rep.

2. End-to-end OTA tests
   These tests will be conducted by manually running python scripts located in `$AFR_HOME/tools/ota_e2e_test/`
   a. Enable the OTA end-to-end tests by setting the following macros to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`:
      - `testrunner_OTA_END_TO_END_ENABLED`
   b. Follow the instructions in `$AFR_HOME/tools/ota_e2e_test/README.md`

Test Execution

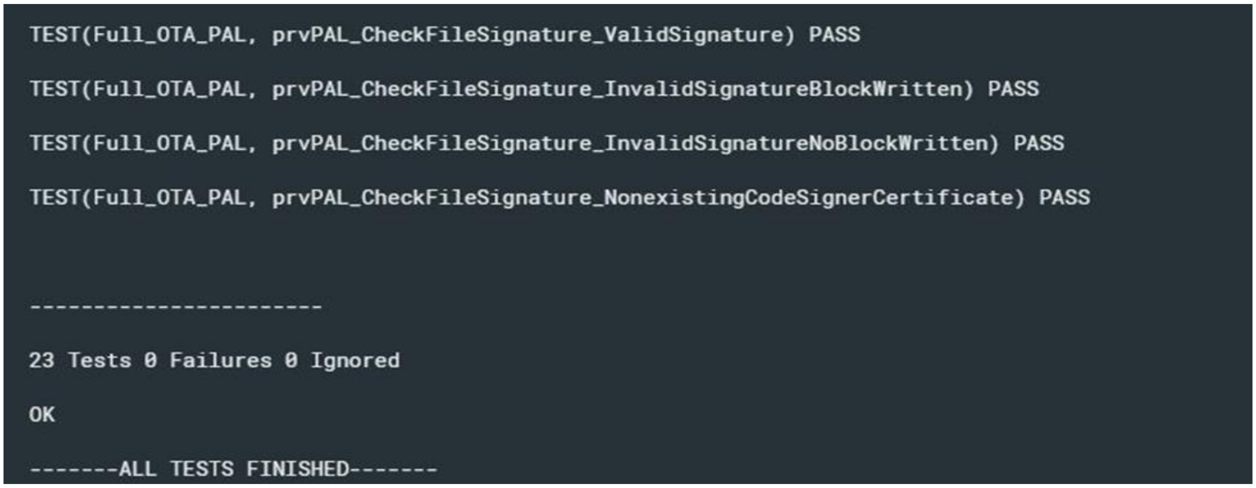
1. Agent/PAL tests
   a. Build and execute the test project.
   b. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file.

Example of the test results output:



```
---------STARTING TESTS---------

TEST(Full_OTA_PAL, prvPAL_CloseFile_ValidSignature) PASS

TEST(Full_OTA_PAL, prvPAL_CloseFile_InvalidSignatureBlockWritten) PASS

TEST(Full_OTA_PAL, prvPAL_CloseFile_InvalidSignatureNoBlockWritten) PASS

TEST(Full_OTA_PAL, prvPAL_CloseFile_NonexistingCodeSignerCertificate) PASS

TEST(Full_OTA_PAL, prvPAL_CreateFileForRx_CreateAnyFile) PASS
```

…



```
TEST(Full_OTA_PAL, prvPAL_CheckFileSignature_ValidSignature) PASS

TEST(Full_OTA_PAL, prvPAL_CheckFileSignature_InvalidSignatureBlockWritten) PASS

TEST(Full_OTA_PAL, prvPAL_CheckFileSignature_InvalidSignatureNoBlockWritten) PASS

TEST(Full_OTA_PAL, prvPAL_CheckFileSignature_NonexistingCodeSignerCertificate) PASS



-----------------------

23 Tests 0 Failures 0 Ignored

OK

-------ALL TESTS FINISHED-------
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

2. End-to-end OTA Tests
    a. Make sure there are no changes to `aws_demo_runner.c`, `aws_clientcredential.h`, `aws_clientcredential_keys.h`, `aws_application_version.h`, and `aws_ota_codesigner_certificate.h` from what was used to run the agent/PAL tests.
    b. Follow the example listed in `$AFR_HOME/tools/ota_e2e_test/README.md` to run the ota end-to-end test script.

Alternatively, you can use AWS IoT Device Tester for Amazon FreeRTOS to test OTA.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.10 Appendix J: Bootloader

Amazon FreeRTOS provides a demo bootloader as an example for the Microchip Curiosity PIC32MZEF platform. It may be ported to other platforms where applicable. However there is the option for users to provide their own bootloader. In order for such a bootloader to work with the OTA functionality, the following requirements must be adhered to:

1. The bootloader shall be stored in non-volatile memory so it cannot be overwritten.
2. The bootloader shall verify the cryptographic signature of the downloaded application image. Signature verification must be consistent with the OTA image signer. See Appendix I: OTA Updates for supported signatures.
3. The bootloader shall not allow rolling back to a previously installed application image.
4. The bootloader shall maintain at least one image that can be booted.
5. If the MCU contains more than one image then the image that is executed shall be the latest (newest). The newest version can be determined based on implementation, for example a user defined sequence number, application version etc. As per other requirements, this can only be the case until a newer version has been verified and proven functional.
6. If the MCU cannot verify any images then it shall place itself into a controlled benign state. In this state it prevents itself from being taken over by ensuring no actions are performed.
7. These requirements shall not be breached even in the presence of an accidental or malicious write to any MCU memory location (key store, program memory, RAM, etc.)
8. The bootloader shall support self-test of a new OTA image. If test execution fails, the bootloader shall roll back to the previous valid image. If test execution succeeds, the image shall be marked valid and the previous version erased.

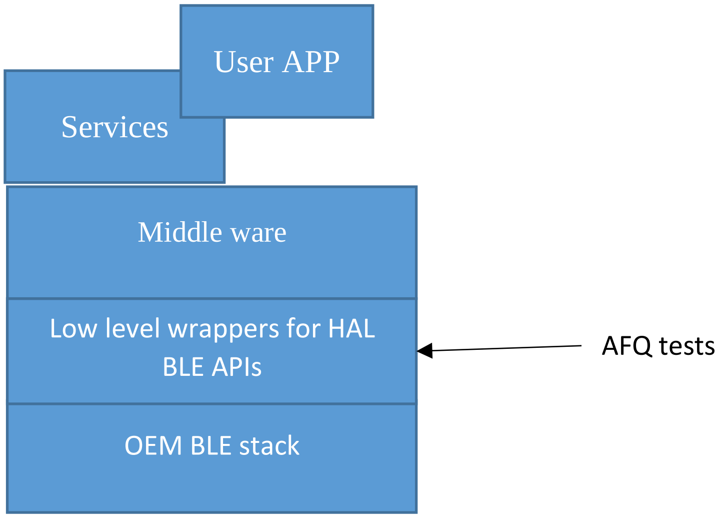
The state of the application must be set by the OTA PAL as described in the user documentation at https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ota-dev.html

5.11 Appendix K: Bluetooth Low Energy (Beta)

Description

The Amazon FreeRTOS Bluetooth Low Energy (BLE) feature allows you to do WIFI provisioning and MQTT over BLE. It also provides users with higher level API that streamline the use of the BLE stack.

The AFR Bluetooth Library is layered as show below:



The AFQ tests target the wrapper layer, just above the Bluetooth Low Energy stack provided by the OEMs. Passing the AFQ tests should ensure the connectivity over BLE is working properly. As for how to use the Amazon FreeRTOS BLE library in your application, see more information here: https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ble-library.html

This library is in beta. Please download the code from GitHub branch https://github.com/aws/amazon-freertos/tree/feature/ble-beta.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied BLE drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.
4. A Raspberry PI 3b+ device.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Preparing the IDE Project

1. Add all files in `$AFR_HOME/lib/bluetooth_low_energy/portable/[vendor]/[board]/` directory to the `[project_top_level]/lib/aws/bluetooth_low_energy` virtual folder of the test IDE project.
2. Add the `$AFR_HOME/lib/include/bluetooth_low_energy` directory to the `[project_top_level]/lib/aws/include` folder of the test IDE project.
3. Add the file `$AFR_HOME/tests/common/ble/aws_test_ble.c` to the `[project_top_level]/application_code/common_tests/ble` virtual folder of the IDE project.
4. Enable the necessary BLE drivers in the `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` driver initializations of the relevant projects in the test project

Porting

The API for BLE feature are defined in three files at `$AFR_HOME/lib/include/bluetooth_low_energy` directory.

- `bt_hal_manager.h`
- `bt_hal_manager_adapter_ble.h`
- `bt_hal_gatt_server.h`

See the comments in the files for description of APIs. The API that MUST be implemented are listed below.

GAP Common (bt_hal_manager.h)		
pxBtManagerInit	pxEnable	pxDisable
pxGetDeviceProperty	pxSetDeviceProperty (All options mandatory except eBTpropertyRemoteRssi, eBTpropertyRemoteVersionInfo)	pxPair
	Y(All options mandatory expect eBTpropertyRemoteRssi, eBTpropertyRemoteVersionInfo)	
pxRemoveBond	pxGetConnectionState	pxPinReply
pxSspReply	pxGetTxpower	pxGetLeAdapter
pxDeviceStateChangedCb	pxAdapterPropertiesCb	pxSspRequestCb
pxPairingStateChangedCb	pxTxPowerCb	

GAP BLE (bt_hal_manager_adapter_ble.h)		
pxRegisterBleApp	pxUnregisterBleApp	pxBleAdapterInit
pxStartAdv	pxStopAdv	pxSetAdvData
pxConnParameterUpdateRequest	pxRegisterBleAdapterCb	pxAdvStartCb

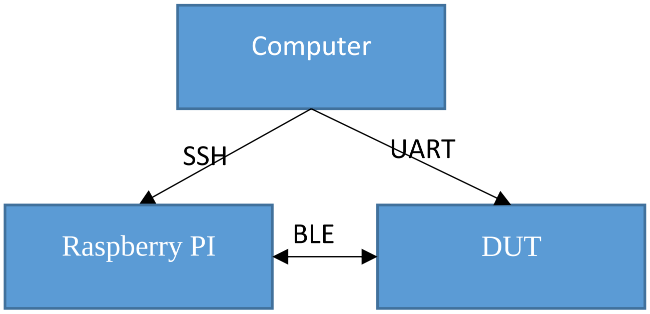
Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

pxSetAdvDataCb	pxConnParameterUpdateRequestCb	pxCongestionCb

GATT Server (bt_hal_gatt_server.h)		
pxRegisterServer	pxUnregisterServer	pxGattServerInit
pxAddService	pxAddIncludedService	pxAddCharacteristic
pxSetVal	pxAddDescriptor	pxStartService
pxStopService	pxDeleteService	pxSendIndication
pxSendResponse	pxMtuChangedCb	pxCongestionCb
pxIndicationSentCb	pxRequestExecWriteCb	pxRequestWriteCb
pxRequestReadCb	pxServiceDeletedCb	pxServiceStoppedCb
pxServiceStartedCb	pxDescriptorAddedCb	pxSetValCallbackCb
pxCharacteristicAddedCb	pxIncludedServiceAddedCb	pxServiceAddedCb
pxConnectionCb	pxUnregisterServerCb	pxRegisterServerCb

Test Setup

The setup for BLE AFQ tests requires a Raspberry PI 3b+ as an external device to run the BLE tests.



The test computer sends test python file to the Raspberry and execute them remotely through ssh. Test results are returned through the ssh client.

At the same time, the test computer runs the tests on the DUT. Tests results are returned thought UART.

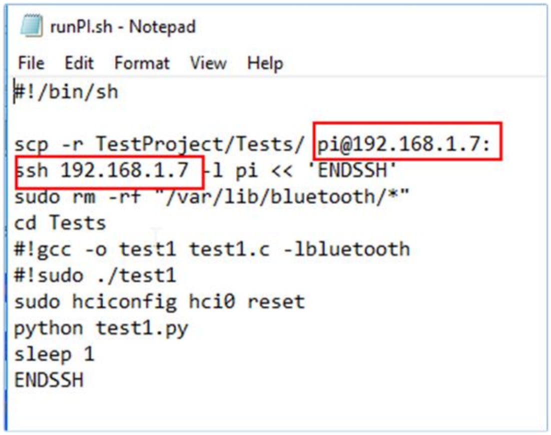
On the Raspberry PI:

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

1. Make sure to order a Raspberry PI 3b+. Previous versions of devices don't have Bluetooth support. You also need to order a memory card with it as Raspberry PI are not provided with a hard drive.
2. Raspberry PI 3b+: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
3. Follow the step here to setup the Raspberry PI with Raspbian OS: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up
4. Download bluez 5.50: https://git.kernel.org/pub/scm/bluetooth/bluez.git
5. Follow the README at the root to install on the Raspberry PI.
6. Enable SSH on the PI: https://www.raspberrypi.org/documentation/remote-access/ssh/

On the test computer:

1. Temporarily (do not push the change) modify the script in `$AFR_HOME/tests/common/framework/bleTestsScripts/runPI.sh` with the IP address of your Raspberry PI:

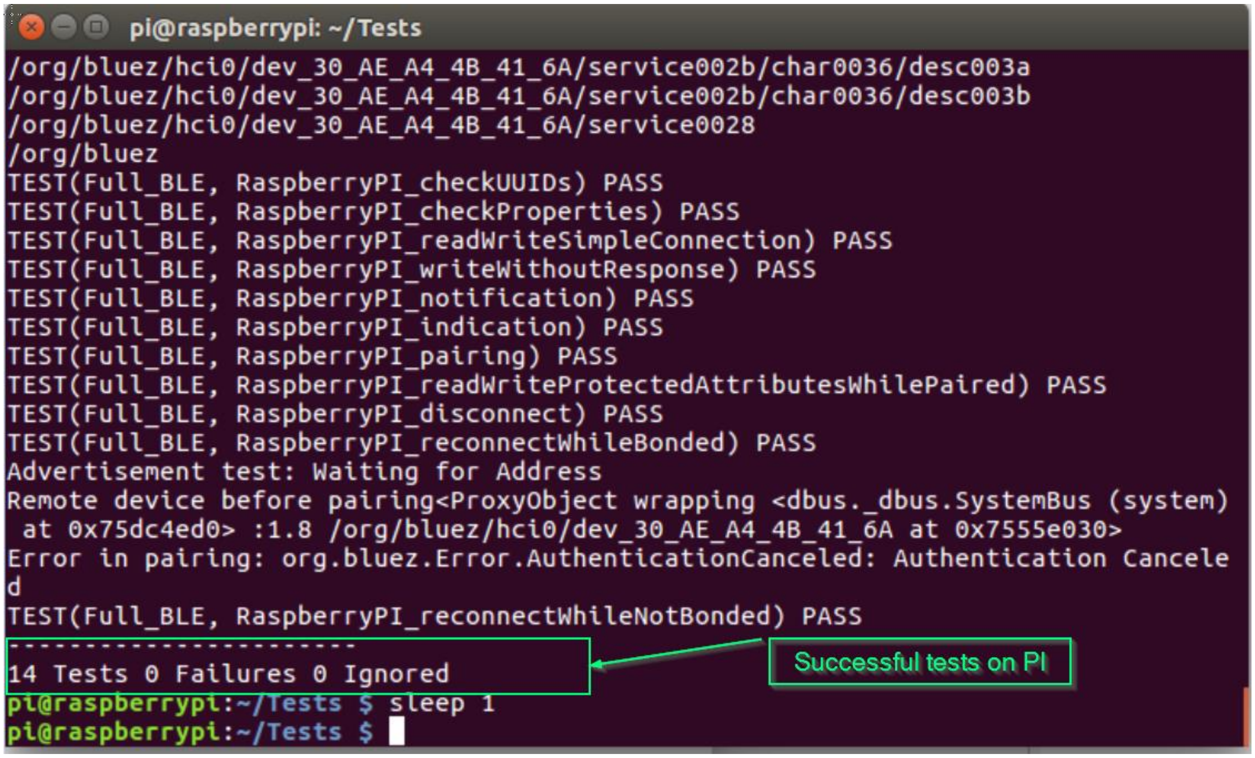


```
#!/bin/sh

scp -r TestProject/Tests/ pi@192.168.1.7:
ssh 192.168.1.7 -l pi << 'ENDSSH'
sudo rm -rf "/var/lib/bluetooth/*"
cd Tests
#!gcc -o test1 test1.c -lbluetooth
#!sudo ./test1
sudo hciconfig hci0 reset
python test1.py
sleep 1
ENDSSH
```

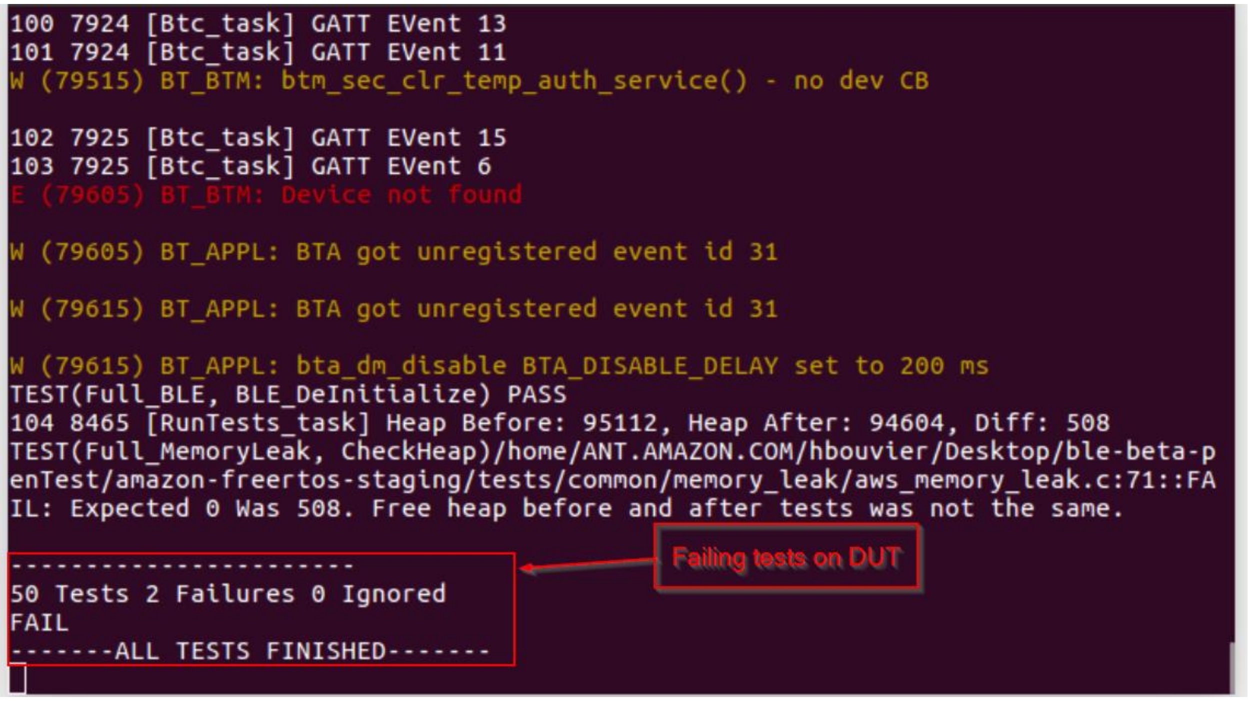
Test Execution

Launch the script runPI.sh and launch the test project. The tests must pass in the PI and on the DUT:

Example of tests results:



```
/org/bluez/hci0/dev_30_AE_A4_4B_41_6A/service002b/char0036/desc003a
/org/bluez/hci0/dev_30_AE_A4_4B_41_6A/service002b/char0036/desc003b
/org/bluez/hci0/dev_30_AE_A4_4B_41_6A/service0028
/org/bluez
TEST(Full_BLE, RaspberryPI_checkUUIDs) PASS
TEST(Full_BLE, RaspberryPI_checkProperties) PASS
TEST(Full_BLE, RaspberryPI_readWriteSimpleConnection) PASS
TEST(Full_BLE, RaspberryPI_writeWithoutResponse) PASS
TEST(Full_BLE, RaspberryPI_notification) PASS
TEST(Full_BLE, RaspberryPI_indication) PASS
TEST(Full_BLE, RaspberryPI_pairing) PASS
TEST(Full_BLE, RaspberryPI_readWriteProtectedAttributesWhilePaired) PASS
TEST(Full_BLE, RaspberryPI_disconnect) PASS
TEST(Full_BLE, RaspberryPI_reconnectWhileBonded) PASS
Advertisement test: Waiting for Address
Remote device before pairing<ProxyObject wrapping <dbus._dbus.SystemBus (system)
 at 0x75dc4ed0> :1.8 /org/bluez/hci0/dev_30_AE_A4_4B_41_6A at 0x7555e030>
Error in pairing: org.bluez.Error.AuthenticationCanceled: Authentication Cancele
d
TEST(Full_BLE, RaspberryPI_reconnectWhileNotBonded) PASS
-----------------------
14 Tests 0 Failures 0 Ignored
pi@raspberrypi:~/Tests $ sleep 1
pi@raspberrypi:~/Tests $ 
```



```
100 7924 [Btc_task] GATT EVent 13
101 7924 [Btc_task] GATT EVent 11
W (79515) BT_BTM: btm_sec_clr_temp_auth_service() - no dev CB

102 7925 [Btc_task] GATT EVent 15
103 7925 [Btc_task] GATT EVent 6
E (79605) BT_BTM: Device not found

W (79605) BT_APPL: BTA got unregistered event id 31

W (79615) BT_APPL: BTA got unregistered event id 31

W (79615) BT_APPL: bta_dm_disable BTA_DISABLE_DELAY set to 200 ms
TEST(Full_BLE, BLE_DeInitialize) PASS
104 8465 [RunTests_task] Heap Before: 95112, Heap After: 94604, Diff: 508
TEST(Full_MemoryLeak, CheckHeap)/home/ANT.AMAZON.COM/hbouvier/Desktop/ble-beta-p
enTest/amazon-freertos-staging/tests/common/memory_leak/aws_memory_leak.c:71::FA
IL: Expected 0 Was 508. Free heap before and after tests was not the same.

-----------------------
50 Tests 2 Failures 0 Ignored
FAIL
-------ALL TESTS FINISHED-------
```

5.12 Appendix L: Test List

The tests listed here are the current tests we provide. It is subject to change.

Library	Tests	Notes
Wi-Fi	WiFiOnOff	
	WiFiMode	
	WiFiConnectionLoop	
	WiFiIsConnected	
	WiFiNetworkAddGetDelete	
	WiFiPowerManagementMode	
	WiFiGetIP	
	WiFiGetMAC	
	WiFiGetHostIP	
	WiFiScan	
	WiFiReset	
	WiFiPing	
	WiFiConnectMultipleAP	
	WiFiSeperateTasksConnectingAndDisconnectingAtOnce	
	WiFiOnOffLoop	
	WIFI_GetMode_NullParameters	
	WIFI_GetIP_NullParameters	
	WIFI_GetMAC_NullParameters	
	WIFI_GetHostIP_NullParameters	
	WIFI_Scan_NullParameters	
	WIFI_NetworkAdd_NullParameters	
	WIFI_NetworkGet_NullParameters	
	WIFI_SetPMMode_NullParameters	
	WIFI_GetPMMode_NullParameters	
	WIFI_Ping_NullParameters	
	WIFI_ConnectAP_NullParameters	
	WIFI_SetMode_InvalidMode	
	WIFI_GetHostIP_InvalidDomainName	
	WIFI_GetHostIP_DomainNameLengthExceeded	
	WIFI_NetworkDelete_DeleteNonExistingNetwork	
	WIFI_NetworkGetNonExistingNetwork	
	WIFI_SetPMMode_InvalidPMMode	
	WIFI_Ping_ZeroParameters	
	WIFI_ConnectAP_InvalidSSID	
	WIFI_ConnectAP_InvalidPassword	
	WIFI_ConnectAP_InvalidSecurityTypes	
	WIFI_ConnectAP_MaxSSIDLengthExceeded	
	WIFI_ConnectAP_MaxPasswordLengthExceeded	
	WIFI_ConnectAP_ZeroLengthSSID	
	WIFI_ConnectAP_ZeroLengthPassword	
	WIFI_ConnectAP_PasswordLengthLess	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Library	Tests	Notes
	WIFI_Scan_ZeroScanNumber	
	WIFI_NetworkGet_GetManyNetworks	
	WIFI_NetworkAdd_AddManyNetworks	
	WIFI_NetworkDelete_DeleteManyNetworks	
	WIFI_ConnectAP_ConnectAllChannels	
Secure Sockets	SOCKETS_Threadsafe_SameSocketDifferentTasks	
	SOCKETS_Threadsafe_DifferentSocketsDifferentTasks	
	SOCKETS_Connect_InvalidAddressLength	
	SOCKETS_Connect_InvalidParams	
	SOCKETS_Socket_TCP	
	SOCKETS_SetSockOpt_RCVTIMEO	
	SOCKETS_SetSockOpt_InvalidParams	
	SOCKETS_Shutdown	
	SOCKETS_ShutdownInvalidParams	
	SOCKETS_ShutdownWithoutReceiving	
	SOCKETS_Close	
	SOCKETS_CloseInvalidParams	
	SOCKETS_CloseWithoutReceiving	
	SOCKETS_Recv_ByteByByte	
	SOCKETS_Recv_On_Unconnected_socket	
	SOCKETS_SendRecv_VaryLength	
	SOCKETS_Socket_InvalidTooManySockets	
	SOCKETS_Socket_InvalidInputParams	
	SOCKETS_Send_Invalid	
	SOCKETS_Recv_Invalid	
	SOCKETS_htos_HappyCase	
	SOCKETS_inet_addr_quick_HappyCase	
	SOCKETS_NonBlocking_Test	
	SECURE_SOCKETS_Threadsafe_DifferentSocketsDifferentTasks	
	SECURE_SOCKETS_Threadsafe_SameSocketDifferentTasks	
	SECURE_SOCKETS_Connect_InvalidAddressLength	
	SECURE_SOCKETS_Connect_InvalidParams	
	SECURE_SOCKETS_NonBlockingConnect	
	SECURE_SOCKETS_NonBlocking_Test	
	SECURE_SOCKETS_SetSockOpt_SERVER_NAME_INDICATION	
	SECURE_SOCKETS_SetSockOpt_TRUSTED_SERVER_CERTIFICATE	
	SECURE_SOCKETS_SetSockOpt_RCVTIMEO	
	SECURE_SOCKETS_SetSockOpt_InvalidParams	
	SECURE_SOCKETS_Shutdown	
	SECURE_SOCKETS_ShutdownInvalidParams	
	SECURE_SOCKETS_ShutdownWithoutReceiving	
	SECURE_SOCKETS_Close	
	SECURE_SOCKETS_CloseInvalidParams	
	SECURE_SOCKETS_CloseWithoutReceiving	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Library	Tests	Notes
	SECURE_SOCKETS_Recv_ByteByByte	
	SECURE_SOCKETS_Recv_On_Unconnected_socket	
	SECURE_SOCKETS_SendRecv_VaryLength	
	SECURE_SOCKETS_SockEventHandler	
	SECURE_SOCKETS_Send_Invalid	
	SECURE_SOCKETS_SetSecureOptionsAfterConnect	
	SECURE_SOCKETS_TwoSecureConnections	
	SECURE_SOCKETS_Recv_Invalid	
TLS	TLS_ConnectEC	
	TLS_ConnectRSA	
	TLS_ConnectMalformedCert	
	TLS_ConnectUntrustedCert	
	TLS_ConnectBYOCCredentials	
PKCS #11	CreateObject_InvalidParams	
	Digest	
	Digest_ErrorConditions	
	FindObjectsFinal_InvalidParams	
	FindObjectsInit_InvalidParams	
	FindObjects_InvalidParams	
	GenerateRandom_HappyPath	
	GenerateRandom_InvalidParams	
	GetAttributeValue_InvalidParams	
	GetFunctionListInvalidParams	
	GetSlotListInvalidParams	
	InitializeFinalizeInvalidParams	
	KeyGenerationEcdsaHappyPath	
	Objects_HappyPath	
	OpenCloseSessionInvalidParams	
	SignInit_InvalidParams	
	SignVerifyCryptoApiInteropRSA	
	SignVerifyRoundTripWithCorrectECPublicKey	
	SignVerifyRoundTripWithCorrectRSAPublicKey	
	SignVerifyRoundTripWithWrongECPublicKey	
	SignVerifyRoundTripWithWrongRSAPublicKey	
	Sign_HappyPath	
	Sign_InvalidParams	
	TestECDSAExport	
	TestECDSAParse	
	TestRSAExport	
	TestRSAParse	
	VerifyInit_InvalidParams	
	Verify_HappyPath	
	Verify_InvalidParams	
MQTT	MQTT_Init_HappyCase	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Library	Tests	Notes
	MQTT_Init_NULLParams	
	MQTT_Connect_HappyCase	
	MQTT_Connect_BrokerRejectsConnection	
	MQTT_Connect_ConnACKWithoutConnect	
	MQTT_Connect_ReservedReturnCodeFromBroker	
	MQTT_Connect_ShorterConnACK	
	MQTT_Connect_LongerConnACK	
	MQTT_Connect_NULLParams	
	MQTT_Connect_SecondConnectWhileAlreadyConnected	
	MQTT_Connect_SecondConnectWhileWaitingForConnACK	
	MQTT_Connect_NetworkSendFailed	
	prvGetTopicFilterType_HappyCases	
	prvGetTopicFilterType_ErrorCases	
	prvDoesTopicMatchTopicFilter_MatchCases	
	prvDoesTopicMatchTopicFilter_NotMatchCases	
OTA	OTA_SetImageState_InvalidParams	
	prvParseJobDocFromJSONandPrvOTA_Close	
	prvParseJSONbyModel_Errors	
	prvPAL_CloseFile_ValidSignature	
	prvPAL_CloseFile_InvalidSignatureBlockWritten	
	prvPAL_CloseFile_InvalidSignatureNoBlockWritten	
	prvPAL_CloseFile_NonexistingCodeSignerCertificate	
	prvPAL_CreateFileForRx_CreateAnyFile	
	prvPAL_Abort_OpenFile	
	prvPAL_Abort_FileWithBlockWritten	
	prvPAL_Abort_NullFileHandle	
	prvPAL_Abort_NonExistentFile	
	prvPAL_WriteBlock_WriteSingleByte	
	prvPAL_WriteBlock_WriteManyBlocks	
	prvPAL_SetPlatformImageState_SelfTestImageState	
	prvPAL_SetPlatformImageState_InvalidImageState	
	prvPAL_SetPlatformImageState_UnknownImageState	
	prvPAL_SetPlatformImageState_RejectImageState	
	prvPAL_GetPlatformImageState_InvalidImageStateFromFileCloseFailure	
	prvPAL_ReadAndAssumeCertificate_ExistingFile	
	prvPAL_CheckFileSignature_ValidSignature	
	prvPAL_CheckFileSignature_InvalidSignatureBlockWritten	
	prvPAL_CheckFileSignature_InvalidSignatureNoBlockWritten	
	prvPAL_CheckFileSignature_NonexistingCodeSignerCertificate	
	OtaTestGreaterVersion	
	OtaTestUnsignedImage	
	OtaTestSameVersion	
	OtaTestUntrustedCertificate	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Library	Tests	Notes
	OtaTestCorruptImageBeforeSigning	
	OtaTestPreviousVersion	
	OtaTestCorruptImageAfterSigning	
	OtaTestCorruptSignature	
	OtaTestSingleByteImage	
	OtaTestMissingFilename	
	OtaTestIncorrectPlatform	
	OtaTestBackToBackDownloads	
	OtaTestIncorrectWifiPassword	
BLE **(test on DUT)**	BLE_Initialize_common_GAP	
	BLE_Initialize_BLE_GAP	
	BLE_Initialize_BLE_GATT	
	BLE_CreateAttTable_CreateServices	
	BLE_CreateAttTable_CreateCharacteristics	
	BLE_CreateAttTable_Descriptors	
	BLE_CreateAttTable_IncludedService	Disabled
	BLE_CreateAttTable_StartService	
	BLE_Advertising_SetProperties	
	BLE_Advertising_SetAvertisementData	
	BLE_Advertising_StartAdvertisement	
	BLE_Connection_SimpleConnection	
	BLE_Connection_UpdateConnectionParamReq	
	BLE_Property_WriteCharacteristic	
	BLE_Property_WriteDescriptor	
	BLE_Property_ReadCharacteristic	
	BLE_Property_ReadDescriptor	
	BLE_Property_WriteNoResponse	
	BLE_Property_Notification	
	BLE_Property_Indication	
	BLE_Connection_Mode1Level4	
	BLE_Connection_Mode1Level4_Property_WriteDescr	
	BLE_Connection_Mode1Level4_Property_WriteChar	
	BLE_Connection_Disconnect	
	BLE_Connection_BondedReconnectAndPair	
	BLE_Connection_Disconnect	
	BLE_Connection_CheckBonding	
	BLE_Connection_RemoveBonding	
	BLE_Connection_Mode1Level2	
	BLE_DeInitialize	
BLE **(test on PI)**	advertisement	
	discoverPrimaryServices	
	simpleConnection	
	checkProperties	
	checkUUIDs	

Library	Tests	Notes
	readWriteSimpleConnection	
	writeWithoutResponse	
	notification	
	indication	
	readWriteProtectedAttributesWhileNotPaired	
	readWriteProtectedAttributesWhilePaired	
	pairing	
	disconnect	
	reconnectWhileBonded	
	reconnectWhileNotBonded	
Total	215	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.13 Appendix M: TLS Server Setup

A simple TLS echo server is provided with Amazon FreeRTOS code. It is located in `$AFR_HOME/tools/echo_server/tls_echo_server.go`.

Instructions:

1. Install the latest version of GO on your server host: https://golang.org/dl/
2. Install openssl on your server host:
    a. Linux --- https://www.openssl.org/source/
    b. Windows --- https://slproweb.com/products/Win32OpenSSL.html
3. Copy `tls_echo_server.go` to a directory you choose.
4. Generate a TLS server self-signed certificate and private key. See `$AFR_HOME/tools/echo_server/readme-gencert.txt` for the openssl commands to generate a self-signed server certificate and private key.
5. Copy the certificate and private key .pem files into a subdirectory called "certs". The "certs" directory should be a subdirectory of the directory where the server code will run.
6. Start the TLS server by running: `go run tls_echo_server.go`
7. The server will listen on port 9000. The IP address and the port must be set in `$AFR_HOME/tests/common/include/aws_test_tcp.h`. For example if your server's IP address is 192.168.2.6, set the following macros:

Macro definition for TLS server	Example value if address is 192.168.0.200
tcptestECHO_SERVER_TLS_ADDR0	192
tcptestECHO_SERVER_TLS_ADDR1	168
tcptestECHO_SERVER_TLS_ADDR2	2
tcptestECHO_SERVER_TLS_ADDR3	6
tcptestECHO_PORT_TLS	(9000)

8. The tests will check the server certificate. In `$AFR_HOME/tests/common/include/aws_test_tcp.h, set tcptestECHO_HOST_ROOT_CA` to your formatted server certificate.
   You can use the formatting tool to format your server certificate.
9. The AFQ secure sockets tests require TLS mutual authentication to be configured. The readme-gencert.txt file also describes how to generate a client certificate and private key that is signed by the server key. This will allow the custom echo server to trust the client certificate presented by your device during TLS authentication. The client certificate and private key must be PEM formatted and copied into aws_clientcredential_keys.h before building and running the test project on the device.

5.14 Appendix N: “Hello World” Demo Project Set Up

Amazon FreeRTOS Directory Structure

All qualified Amazon FreeRTOS ports use the same directory structure, so all new files, including IDE project files, must be created in the correct folder locations. The directory structure is explained below.

The three root level folders under `$AFR_HOME` are:

```
$AFR_HOME
├───demos    Contains projects that build demo applications
├───lib      Contains Amazon FreeRTOS and third-party libraries
└───tests    Contains projects that build qualification tests
```

Your project is to be created within the demos folder, which is structured as follows:

```
$AFR_HOME
└───demos
    ├───common     Contains files built by all demo projects
    ├───pc         Contains a demo project for the FreeRTOS Windows port
    └───vendor     Contains your vendor specific code
        └───board  Contains your board specific code
```

The `$AFR_HOME/demos/[vendor]/[board]` folder is a template provided to simplify the creation of a new test project and ensures all test projects have a consistent organization. It has the following structure:

```
$AFR_HOME
└───demos
    └───[vendor]
        └───[board]
            ├───common
            │   ├───application_code   Contains main.c
            │   │   └───vendor_code    Contains vendor supplied board specific files
            │   └───config_files       Contains Amazon FreeRTOS config files
            └───ide                    Contains an IDE specific project
```

Your demo projects always require vendor-supplied driver libraries. Some vendor-supplied libraries, such as a header file that maps a GPIO output to an LED, are specific to the target development board. Other vendor-supplied libraries, such as the GPIO library itself, are specific to the target MCU family. Do not save vendor-supplied libraries that are specific to the MCU anywhere within either the `$AFR_HOME/tests or $AFR_HOME/demos` folders.

Preparing Your Project Directories:

1. Rename the `$AFR_HOME/demos/vendor` folder to the name of the company that manufactures the MCU – from here on the folder is referred to as `[vendor]`.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

2. Rename the `$AFR_HOME/demos/[vendor]/board` folder to the name of the development board being qualified – from here on the folder is referred to as `[board]`.
3. Copy your `main.c` and `main.h` in `$AFR_HOME/demos/[vendor]/[board]/common/application_code` folder. You can re-use the `main.c` in your `aws_tests` project.
4. Save any required vendor-supplied libraries that are specific to the board in the `$AFR_HOME/demos/[vendor]/[board]/common/application_code/vendor_code` folder.
5. Rename the `$AFR_HOME/demos/[vendor]/[board]/ide` folder to the name of the IDE that will be used to build the test project – from here on the folder is referenced as `[ide]`.

Create the "Hello World" Demo Project

If your IDE does not use relative paths, define a variable in the project for relative folder locations before importing Amazon FreeRTOS source files.

1. Create an IDE project `aws_demos` in the `$AFR_HOME/demos/[vendor]/[board]/[ide]` directory.
2. Create the project structure in the IDE
3. Create three top level virtual folders:
   a. `application_code`
   b. `config_files`
   c. `lib`
4. Import the `$AFR_HOME/demos/[vendor]/[board]/common/application_code` directory and its contents into the `application_code` virtual folder.
5. Import the files in `$AFR_HOME/demos/[vendor]/[board]/common/config_files` into the `config_files` virtual folder.
6. Create a virtual folder under `application_code` and call it `common_demos`.
7. Create a source folder under `common_demos`.
8. Import the files in each of the following directories into the source folder:
   a. `$AFR_HOME/demos/common/demo_runner`
   b. `$AFR_HOME/demos/common/devmode_key_provisioning` (only the .c file)
   c. `$AFR_HOME/demos/common/mqtt`
   d. `$AFR_HOME/demos/common/logging`
9. Import the following directories and its contents into `common_demos` folder.
   a. `$AFR_HOME/demos/common/include`
10. Create two virtual folders `aws` and `third_party` under virtual folder `lib`
11. Import each of the following directories and their contents into the `aws` folder:
    a. `$AFR_HOME/lib/bufferpool`
    b. `$AFR_HOME/lib/FreeRTOS`
    c. `$AFR_HOME/lib/FreeRTOS/portable/MemMang/heap_4.c`
    d. `$AFR_HOME/lib/FreeRTOS/portable/[compiler your IDE uses]`
    e. `$AFR_HOME/lib/FreeRTOS-Plus-TCP` (if you have ported this library, please refer to Appendix D on what files should be included in this project)
    f. `$AFR_HOME/lib/include`
    g. `$AFR_HOME/lib/include/private` (only .h files)
    h. `$AFR_HOME/lib/mqtt`
    i. `$AFR_HOME/lib/pkcs11/portable/[vendor]/[board]/pkcs11.c` (under pkcs11 folder)

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

`$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c` (under secure_sockets folder)

j. `$AFR_HOME/lib/tls` (if you have ported TLS library)

k. `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` (under `wifi` folder, if you have ported WI-FI library)

12. Import each of the following directories and their contents into third_party:
    a. `$AFR_HOME/lib/third_party/mcu_vendor/[vendor]/[board]/[driver_library]/[driver_library_version]` (under mcu_vendor folder)
    b. `$AFR_HOME/lib/third_party/mbedtls` (rename `../mbedtls/library` to `../mbedtls/source`)
    c. `$AFR_HOME/lib/third_party/pkcs11`
13. Make sure the following compiler include paths are set in the project property:
    a. `$AFR_HOME/demos/common/include`
    b. `$AFR_HOME/lib/include`
    c. `$AFR_HOME/lib/include/private`
    d. `$AFR_HOME/lib/FreeRTOS/portable/[compiler]/[architecture]`
    e. `$AFR_HOME/demos/vendor/board/common/config_files`
    f. `$AFR_HOME/lib/third_party/mbedtls/include`
    g. Any paths required by vendor-supplied driver libraries

5.15 Appendix O: Checklist for Qualification

□ "Test" project passed all tests in AFQ tests version ______________.

  - □ Ported configPRINT_STRING() macro.
  - □ Configured FreeRTOS kernel according to your target MCU.
  - □ Ported Wi-Fi Management library (Optional if your board does not support Wi-Fi) and passed Wi-Fi Management library tests.
  - □ Ported OTA library (Optional if your board does not support Wi-Fi) and passed OTA library tests.
  - □ Bootloader following the Amazon FreeRTOS guidelines in Appendix J: Bootloader
  - □ Ported FreeRTOS TCP/IP stack (Optional if you use off-chip TCP/IP stack).
  - □ Ported CYPTO library and passed CYPTO library tests.
  - □ Ported PKCS #11 library and passed tests for this library.
  - □ Ported TLS library (Optional if you use Amazon FreeRTOS TLS support) and passed tests for this library.
  - □ Ported Secure Sockets library and passed the associated tests.
  - □ Passed tests for MQTT library.

□ Prepare a "Demo" project for an IDE you choose that can send "Hello World" to AWS IoT Console and receive reply through MQTT protocol.

□ Put the appropriate open source license text in your code. Please refer to https://opensource.org/licenses for license text information.

□ Configure your board name in

```
$AFR_HOME/demos/[vendor]/[board]/common/config_files/FreeRTOSConfig.h
#define mqttconfigMETRIC_PLATFORM     "Platform=Your board name"
```

□ Information required for Appendix U: Hardware Information filled

□ Prepare a "Getting Started Guide" for your board to help users run your "Demo" project. You can use the Getting Started Guide template to start and look at the guide for the Window Simulator for reference.

□ Provide Appendix V: Information for listing on the Amazon FreeRTOS Console

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.16 Appendix P: Troubleshooting Porting Setup

1. **Can I reach the "echo server" from two different networks (for example, from two subnets across 2 different access points)?**
   An echo server is required for successful completion of the TCP/IP and TLS tests. The echo server must be reachable from the network that the boards are connected to. Please consult your IT support to enable routing across subnets if you need devices on different subnets to communicate to a single echo server.

2. **Can I use openssl in a Windows environment?**
   Yes. Even though only a Linux distribution of openssl is provided on https://www.openssl.org/, you can find openssl distributions for Windows on the internet.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.17 Appendix Q: Instructions to Create a BYOC (ECDSA)

Prerequisite:

To follow the instructions below, you need to have **openssl** and the **AWS CLI** installed.

- **OpenSSL** is an open source toolkit for the TLS protocol. (https://www.openssl.org/). We will use openssl in examples for generating certificates below. Please use TLS V1.2. You can download it here:
  Linux --- https://www.openssl.org/source/
- **AWS** CLI installation guide:
  https://docs.aws.amazon.com/cli/latest/userguide/installing.html
  - MUST DO: Configure AWS CLI before use.
    Please follow the instruction here to configure AWS CLI:
    https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

Note: during the CA certificate creation process, please consider fill in valid information. You may see errors if the organization or other fields don't align in later signing steps.

Generate a Root CA

1. Generate a root CA private key
   a. `openssl ecparam -name prime256v1 -genkey -noout -out rootCA.key`
2. Generate a root CA certificate
   a. `openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.crt`

Generate Intermediate CA

1. Create necessary files
   a. `touch index.txt`
   b. `echo 1000 > serial`
2. Paste the ca.config file in Appendix R: Source for ca.config in the directory
3. Generate intermediate CA's private key:
   a. `openssl ecparam -name prime256v1 -genkey -noout -out intermediateCA.key`
4. Generate intermediate CA's CSR [Make sure to fill Common Name to some value]
   a. `openssl req -new -sha256 -key intermediateCA.key -out intermediateCA.csr`
5. Sign the intermediate CA's CSR with root CA
   b. `openssl ca -config ca.config -notext -cert rootCA.crt -keyfile rootCA.key -days 500 -in  intermediateCA.csr -out intermediateCA.crt`

Generate Device Certificate (ECDSA certificate as an example)

1. Generate private key
   a. `openssl ecparam -name prime256v1 -genkey -noout -out deviceCert.key`
2. Generate CSR for device certificate
   a. `openssl req -new -key deviceCert.key -out deviceCert.csr`
3. Sign the device certificate with the intermediate CA

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

   a.
   ```
   openssl x509 -req -in deviceCert.csr -CA intermediateCA.crt -CAkey intermediateCA.key -CAcreateserial -out deviceCert.crt -days 500 -sha256
   ```

Register both CA certificates

1. Get registration code
   a.
   ```
   aws iot get-registration-code
   ```
2. Generate private key for verification certificates
   a.
   ```
   openssl ecparam -name prime256v1 -genkey -noout -out verificationCert.key
   ```
3. Create CSR for verification certificates. **Set the Common Name field to your registration code obtained in the first step.**
   a.
   ```
   openssl req -new -key verificationCert.key -out verificationCert.csr
   ```
4. Sign a verification certificate using root CA and another one using intermediate CA
   a.
   ```
   openssl x509 -req -in verificationCert.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out rootCAverificationCert.crt -days 500 -sha256
   ```
   b.
   ```
   openssl x509 -req -in verificationCert.csr -CA intermediateCA.crt -CAkey intermediateCA.key -CAcreateserial -out intermediateCAverificationCert.crt -days 500 -sha256
   ```
5. Register both CA certificates with AWS IoT
   a.
   ```
   aws iot register-ca-certificate --ca-certificate file://rootCA.crt --verification-cert file://rootCAverificationCert.crt
   ```
   b.
   ```
   aws iot register-ca-certificate --ca-certificate file://intermediateCA.crt --verification-cert file://intermediateCAverificationCert.crt
   ```
6. Activate both CA certificates
   a.
   ```
   aws iot update-ca-certificate --certificate-id xxxxxxxxxxxxxxxx --new-status ACTIVE
   ```

Register Device Certificate

1. Register the device certificate with AWS IoT
   a.
   ```
   aws iot register-certificate --certificate-pem file://deviceCert.crt --ca-certificate-pem file://intermediateCA.crt
   ```
2. Activate the device certificate
   a.
   ```
   aws iot update-certificate --certificate-id xxxxxxxxxxxxxx --new-status ACTIVE
   ```

deviceCert.crt is device certificate and deviceCert.key is device private key.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.18 Appendix R: Source for ca.config

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir             = .                     # Where everything is kept
certs           = $dir                  # Where the issued certs are kept
crl_dir         = $dir                  # Where the issued crl are kept
database= $dir/index.txt# database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir                  # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber# the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE= $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
```

```
# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions= crl_ext

default_days    = 365                    # how long to certify for
default_crl_days      = 30                       # how long before next CRL
default_md      = default                # use public key default MD
preserve= no                     # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName              = match
stateOrProvinceName      = match
organizationName= match
organizationalUnitName   = optional
commonName               = supplied
emailAddress             = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName              = optional
stateOrProvinceName      = optional
localityName             = optional
organizationName= optional
organizationalUnitName   = optional
commonName               = supplied
emailAddress             = optional

####################################################################
[ req ]
default_bits             = 2048
default_keyfile = privkey.pem
distinguished_name       = req_distinguished_name
attributes               = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                      = Country Name (2 letter code)
countryName_default              = AU
countryName_min                  = 2
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

```
countryName_max                  = 2

stateOrProvinceName              = State or Province Name (full name)
stateOrProvinceName_default      = Some-State

localityName                     = Locality Name (eg, city)

0.organizationName               = Organization Name (eg, company)
0.organizationName_default       = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
#1.organizationName              = Second Organization Name (eg, company)
#1.organizationName_default      = World Wide Web Pty Ltd

organizationalUnitName           = Organizational Unit Name (eg, section)
#organizationalUnitName_default  =

commonName                       = Common Name (e.g. server FQDN or YOUR name)
commonName_max                   = 64

emailAddress                     = Email Address
emailAddress_max         = 64

# SET-ex3                        = SET extension number 3

[ req_attributes ]
challengePassword                = A challenge password
challengePassword_min            = 4
challengePassword_max            = 20

unstructuredName         = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:TRUE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                     = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                        = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

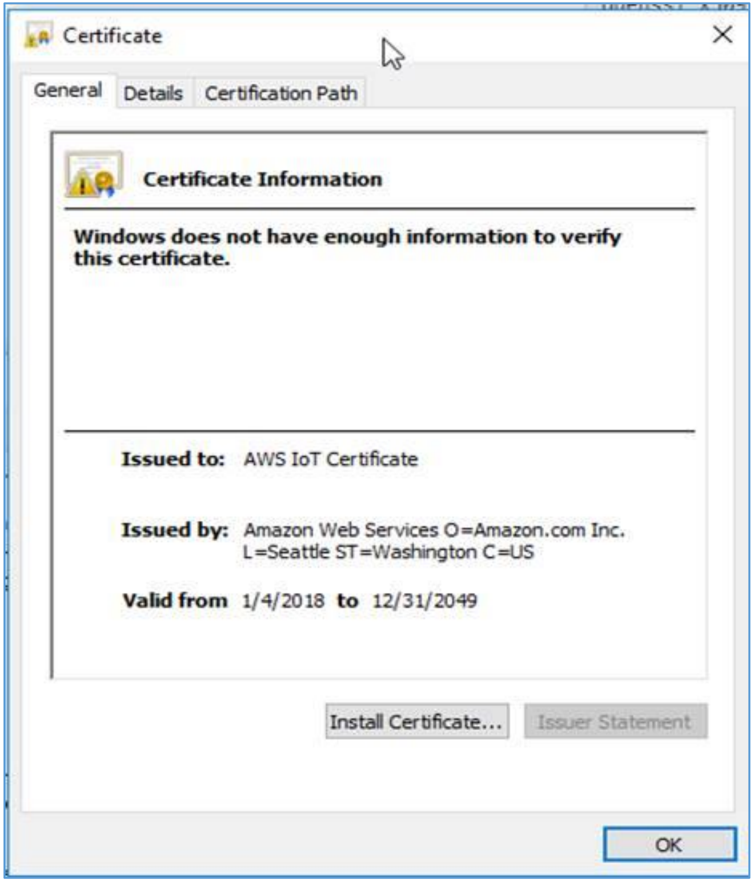
```
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl               = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.
# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                     = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                        = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl               = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]

default_tsa = tsa_config1        # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir             = ./demoCA               # TSA root directory
serial          = $dir/tsaserial # The current serial number (mandatory)
```

```
crypto_device   = builtin               # OpenSSL engine to use for signing
signer_cert     = $dir/tsacert.pem      # The TSA signing certificate
                                        # (optional)
certs           = $dir/cacert.pem       # Certificate chain to include in reply
                                        # (optional)
signer_key      = $dir/private/tsakey.pem # The TSA private key (optional)

default_policy  = tsa_policy1           # Policy if request did not specify it
                                        # (optional)
other_policies  = tsa_policy2, tsa_policy3      # acceptable policies (optional)
digests         = md5, sha1             # Acceptable message digests (mandatory)
accuracy= secs:1, millisecs:500, microsecs:100   # (optional)
clock_precision_digits  = 0     # number of digits after dot. (optional)
ordering        = yes   # Is ordering defined for timestamps?
                                # (optional, default: no)
tsa_name        = yes   # Must the TSA name be included in the reply?
                                # (optional, default: no)
ess_cert_id_chain       = no    # Must the ESS cert id chain be included?
                                # (optional, default: no)
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.19 Appendix S: Modify issuer in a certificate

1. Take the valid client certificate that you have been using as a base. In this example is it 81909ac548-certificate.pem.crt



Certificate

Certificate Information

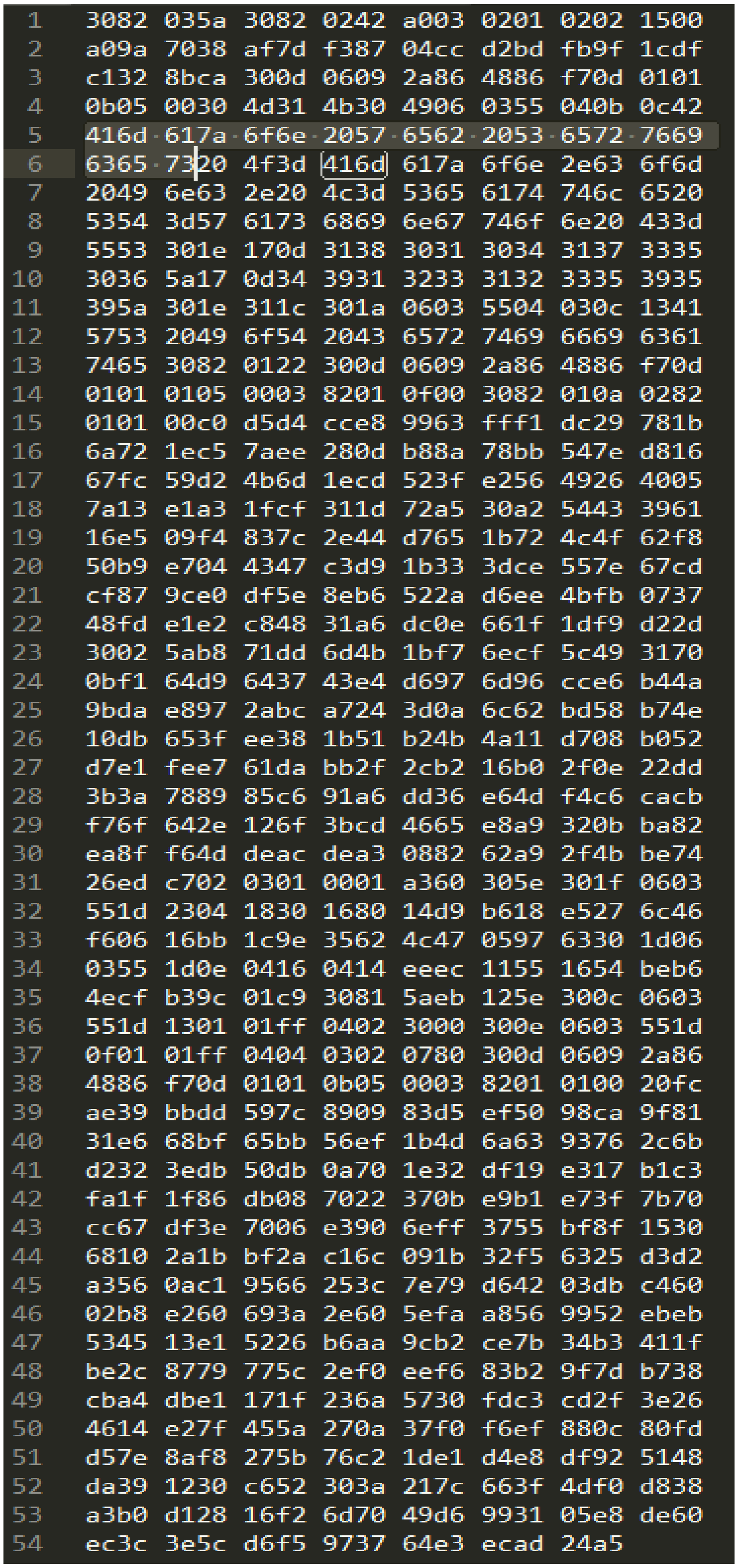
Windows does not have enough information to verify this certificate.

Issued to: AWS IoT Certificate

Issued by: Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US

Valid from 1/4/2018 **to** 12/31/2049

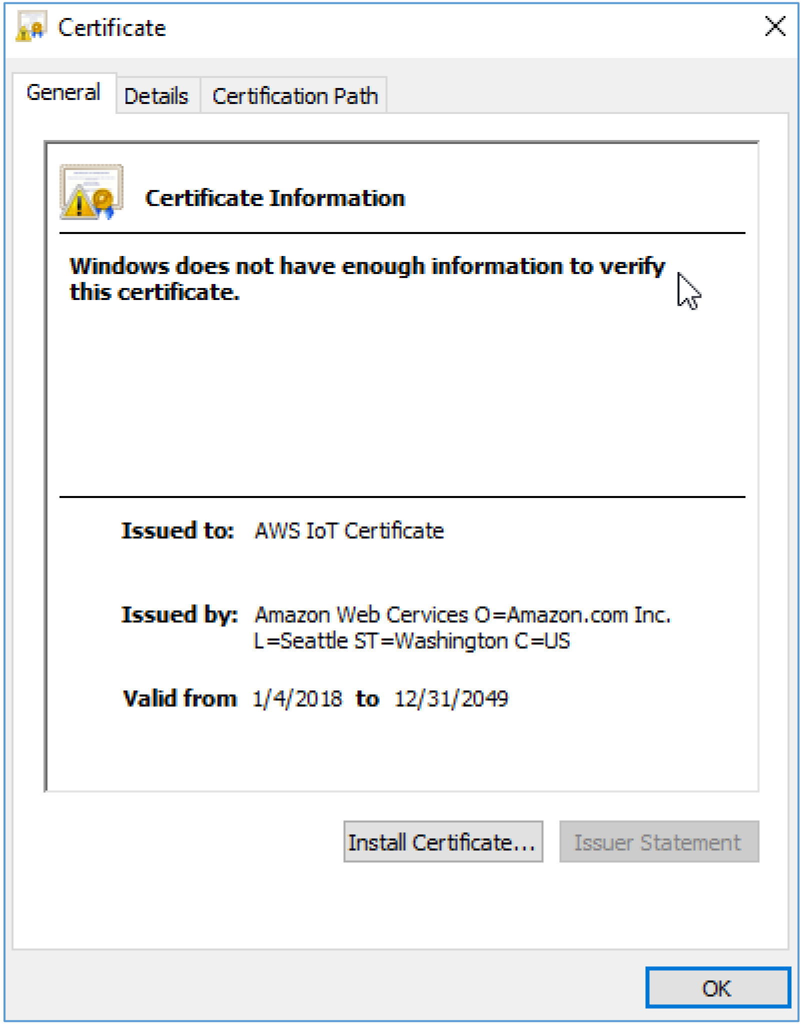
2. Convert the certificate from PEM to DER (openssl x509 –outform der –in 81909ac548-certificate.pem.crt –out 81909ac548-certificate.der.crt)
3. Open the .der certificate. "Amazon Web Services" in hex is **41 6d 61 7a 6f 6e 20 57 65 62 20 53 65 72 76 69 63 65 73**. Search for this sequence in your DER output:



```
3082 035a 3082 0242 a003 0201 0202 1500
a09a 7038 af7d f387 04cc d2bd fb9f 1cdf
c132 8bca 300d 0609 2a86 4886 f70d 0101
0b05 0030 4d31 4b30 4906 0355 040b 0c42
416d 617a 6f6e 2057 6562 2053 6572 7669
6365 7320 4f3d 416d 617a 6f6e 2e63 6f6d
2049 6e63 2e20 4c3d 5365 6174 746c 6520
5354 3d57 6173 6869 6e67 746f 6e20 433d
5553 301e 170d 3138 3031 3034 3137 3335
3036 5a17 0d34 3931 3233 3132 3335 3935
395a 301e 311c 301a 0603 5504 030c 1341
5753 2049 6f54 2043 6572 7469 6669 6361
7465 3082 0122 300d 0609 2a86 4886 f70d
0101 0105 0003 8201 0f00 3082 010a 0282
0101 00c0 d5d4 cce8 9963 fff1 dc29 781b
6a72 1ec5 7aee 280d b88a 78bb 547e d816
67fc 59d2 4b6d 1ecd 523f e256 4926 4005
7a13 e1a3 1fcf 311d 72a5 30a2 5443 3961
16e5 09f4 837c 2e44 d765 1b72 4c4f 62f8
50b9 e704 4347 c3d9 1b33 3dce 557e 67cd
cf87 9ce0 df5e 8eb6 522a d6ee 4bfb 0737
48fd e1e2 c848 31a6 dc0e 661f 1df9 d22d
3002 5ab8 71dd 6d4b 1bf7 6ecf 5c49 3170
0bf1 64d9 6437 43e4 d697 6d96 cce6 b44a
9bda e897 2abc a724 3d0a 6c62 bd58 b74e
10db 653f ee38 1b51 b24b 4a11 d708 b052
d7e1 fee7 61da bb2f 2cb2 16b0 2f0e 22dd
3b3a 7889 85c6 91a6 dd36 e64d f4c6 cacb
f76f 642e 126f 3bcd 4665 e8a9 320b ba82
ea8f f64d deac dea3 0882 62a9 2f4b be74
26ed c702 0301 0001 a360 305e 301f 0603
551d 2304 1830 1680 14d9 b618 e527 6c46
f606 16bb 1c9e 3562 4c47 0597 6330 1d06
0355 1d0e 0416 0414 eeec 1155 1654 beb6
4ecf b39c 01c9 3081 5aeb 125e 300c 0603
551d 1301 01ff 0402 3000 300e 0603 551d
0f01 01ff 0404 0302 0780 300d 0609 2a86
4886 f70d 0101 0b05 0003 8201 0100 20fc
ae39 bbdd 597c 8909 83d5 ef50 98ca 9f81
31e6 68bf 65bb 56ef 1b4d 6a63 9376 2c6b
d232 3edb 50db 0a70 1e32 df19 e317 b1c3
fa1f 1f86 db08 7022 370b e9b1 e73f 7b70
cc67 df3e 7006 e390 6eff 3755 bf8f 1530
6810 2a1b bf2a c16c 091b 32f5 6325 d3d2
a356 0ac1 9566 253c 7e79 d642 03db c460
02b8 e260 693a 2e60 5efa a856 9952 ebeb
5345 13e1 5226 b6aa 9cb2 ce7b 34b3 411f
be2c 8779 775c 2ef0 eef6 83b2 9f7d b738
cba4 dbe1 171f 236a 5730 fdc3 cd2f 3e26
4614 e27f 455a 270a 37f0 f6ef 880c 80fd
d57e 8af8 275b 76c2 1de1 d4e8 df92 5148
da39 1230 c652 303a 217c 663f 4df0 d838
a3b0 d128 16f2 6d70 49d6 9931 05e8 de60
ec3c 3e5c d6f5 9737 64e3 ecad 24a5
```

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

4. Modify the sequence to say 'Amazon Web Cervices', switching out the 53 to be a 43. Save the file. To verify your change, you can check out the modified cert in the windows certificate manager. See that it now says **Issued by**: Amazon Web Cervices



Certificate Information

Windows does not have enough information to verify this certificate.

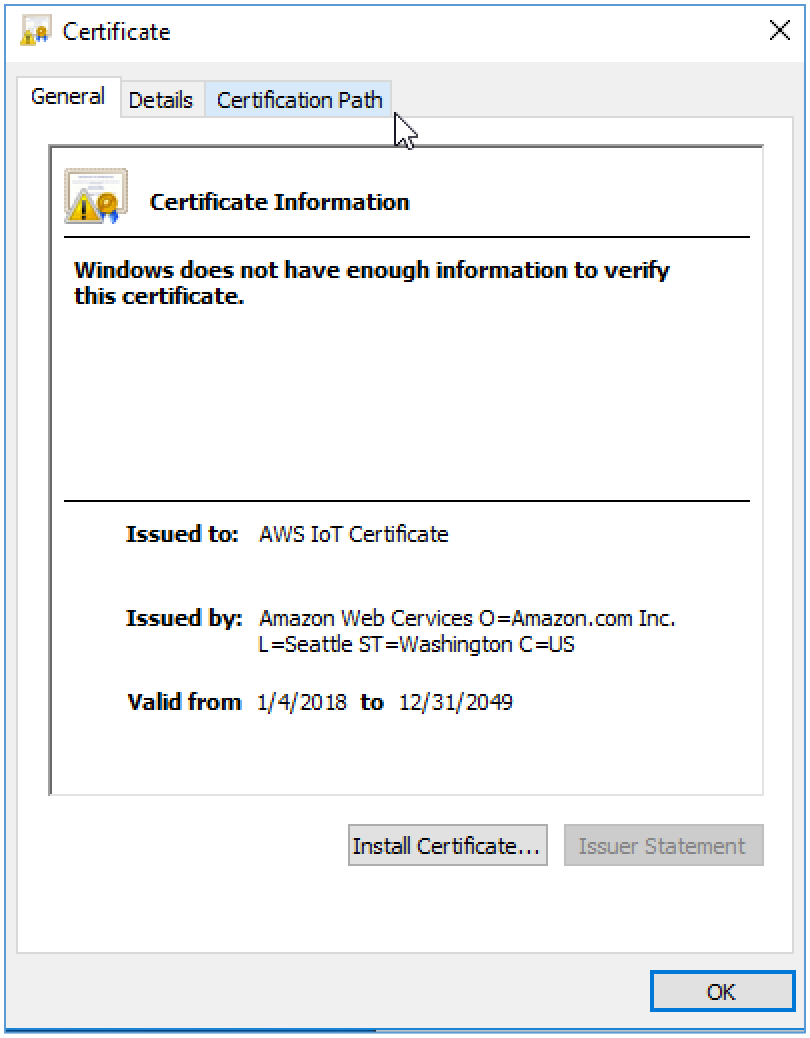
Issued to: AWS IoT Certificate

Issued by: Amazon Web Cervices O=Amazon.com Inc. L=Seattle ST=Washington C=US

Valid from 1/4/2018 **to** 12/31/2049

5. Convert your newly modified certificate back to PEM. openssl x509 –inform der –in 81909ac548–certificate.der.crt –out 81909ac548–cert–modified.pem.crt

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.



Certificate Information

Windows does not have enough information to verify this certificate.

Issued to: AWS IoT Certificate

Issued by: Amazon Web Cervices O=Amazon.com Inc. L=Seattle ST=Washington C=US

Valid from 1/4/2018 **to** 12/31/2049

Again, viewing this in the certificate viewer should show the modified certificate.

6. Put this certificate into the Certificate Configuration Tool ($AFR_HOME\tools\certificate_configuration\CertificateConfigurator.html) and copy the formatted output.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.20 Appendix T: Getting Started Guide Template

Getting Started with the [board-name]

Provide a brief description of the board(s) that are qualified to run Amazon FreeRTOS with links to more in-depth information on your company's website

- What hardware is required?
- What host operating systems are supported?
- What IDEs are supported? (Include links to download IDEs)
- What toolchains will the developer use? (Include links to download toolchains)
- Prerequisite

Prerequisites

List any prerequisites for your board

Setting up the [board-name] Hardware

Provide instructions for setting up the hardware including:

- Jumper settings
- Driver installation (include links to supported driver versions)
- Connecting the board to a computer

Setting Up Your Environment

- Provide instructions to establish a serial connection to your board for each host operating system.
- Provide instructions and link(s) to set up the toolchain for each host operating system.
- Provide instructions for installing/configuring any board-specific software for each host operating system (anything listed here should be called out in the prerequisites section).

Build and Run the FreeRTOS Samples

- Provide instructions for loading/importing the Amazon FreeRTOS sample code into your IDE.
- Provide instructions on how to flash the sample application to your board including:
    - How to connect your board to the host computer
    - How to use an IDE or other tools to flash the sample application to your board
    - How to verify the sample application is running correctly
    - Troubleshooting steps for resolving problems

Debugging the samples

- Provide instructions on how to use any on-board debugging interface or external debuggers for each supported host OS.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.21 Appendix U: Hardware Information

General Information:

Company Name	
Company Name (short, if any) for Amazon FreeRTOS Console	
High Resolution Logo	
Link to Landing Page	
Company Description (15 words)	

Development Board Information:

Board Name	
Board Name (20 chars) for Amazon FreeRTOS Console	
High Resolution Board Image	
Board Description	
Board Description (50 chars) for Amazon FreeRTOS Console	
Microcontroller Family Name	
Board Datasheet	
Compiler Options (optimization)	
IDE with Version Number	
CLI command to build target executables	
CLI command to flash target	
Link to Board Landing Page	
Getting Started Guide	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Link to Purchase Board	

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.22 Appendix V: Information for listing on the Amazon FreeRTOS Console

To list the qualified board(s) on the Amazon FreeRTOS console, we require you to provide additional information based on CMakefile Template. Please follow the guide listed at https://github.com/aws/amazon-freertos/tree/feature/cmake/cmake/doc/porting_guide.md

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

5.23Appendix W: Glossary

$AFR_HOME	The path where Amazon FreeRTOS is installed/extracted.

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.