

# Instruction Manual for the wolfSSL Example Application

Target: Renesas RX72N Envision Kit

RTOS: FreeRTOS+ IoT libraries

# TABLE OF CONTENTS

What Is this document for?	3
Example program structure	3
Requirements for building and running the example project	4
Procedure for creating a wolfSSL example application project	4
1.Create a new executable project	5
Create a new FreeRTOS project	5
2.Device Info Settings	8
3.Adding FIT Modules	8
4.Copy wolfSSL package	13
5. Section setting	13
6.Adding wolfSSL Library and wolfSSL demo files	14
Importing wolfssl library project	15
Adding wolfssl demo application files	15
Adding Include file paths to the project	18
Adding Preprocessor macro	18
Adding Link Library	19
7.Adding wolfSSL demo as a task	19
Execution of the wolfSSL demo application	20
Crypto-test demo	20
Crypto-Benchmark demo	21
TLS-Client demo	22
Things todo when using user's root CA cert	25
Limitations	25
Resources	26
Renesas sites	26
wolfCCL sites	26

Support and Contact
---------------------

# WHAT IS THIS DOCUMENT FOR?

This document is an instruction to add wolfSSL TLS library and to run an example program on the Renesas RX72N Envision Kit. The target MCU is expected to be used with a real-time OS when the product is installed. Therefore, this example program is provided in a configuration that uses FreeRTOS and FreeRTOS + TCP. The steps for generating and executing the program as a new project of e² studio, an IDE made by Renesas, is explained below.

# **EXAMPLE PROGRAM STRUCTURE**

The FreeRTOS kernel and FreeRTOS + TCP protocol stack are required to execute this example program, but they are automatically prepared when creating a new e² studio project. A script that is automatically executed when the project is created, downloads the FreeRTOS-related source files from the GitHub and configures the settings necessary for operation on the evaluation board. The downloaded FreeRTOS IoT Libraries include several demo applications, and the demo application selected from them is configured to be executed.

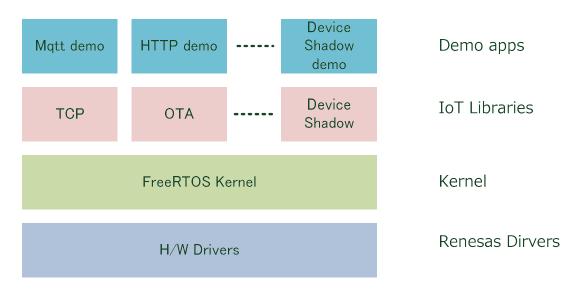


fig.1. Original Structure of the FreeRTOS+IoT libraries demos

The wolfSSL example program adds the wolfSSL library, the wolfSSL demo application, and the FIT components required as the H/W driver to this configuration, and configures it as shown in fig.2.

### Additional Components Device wolfSSL Matt demo HTTP demo Shadow Demo apps demo demo Device **IoT Libraries TCP** OTA wolfSSL Shadow FreeRTOS Kernel Kernel Renesas Dirvers H/W Drivers TSIP Driver

fig.2. The extended structure by adding wolfSSL demo

The added wolfSSL demo application runs as a task on the FreeRTOS kernel and utilizes the TCP protocol stack as a communication channel. In addition, the wolfSSL library supports TSIP. By replacing some of the encryption and TLS functions implemented by the wolfSSL library as software with H/W (TSIP), it is possible to significantly improve the processing speed.

# REQUIREMENTS FOR BUILDING AND RUNNING THE EXAMPLE PROJECT

Tools and components required for the building and execution of this example program:

- 1. e<sup>2</sup> studio Version 2021-10 or later
- 2. CC-RX Tool Chain V3.04 or later
- 3. TSIP v1.15 or later
- 4. RTOS v202002.00-rx-1.0.1 or later
- 5. wolfSSL v5.3.o or later

# PROCEDURE FOR CREATING A WOLFSSL EXAMPLE APPLICATION PROJECT

The following steps are roughly required to execute this example program:

- 1. Create an executable project including FreeRTOS+IoT libraries on e<sup>2</sup>Studio
- 2. Settings for the target MCU and the evaluation board
- 3. Adding FIT components and their update

- 4. Copy wolfSSL package
- 5. Section settings
- 6. Adding wolfSSL library project and wolfSSL demo files
- 7. Execution of wolfSSL demo

From now on, the above steps will be described in that order.

### 1. CREATE A NEW EXECUTABLE PROJECT

Launch e<sup>2</sup> studio and specify a folder to be its workspace. The folder will be the base folder of the project. Here after, the folder is referred as **<base>** in this document.

### CREATE A NEW FREERTOS PROJECT

Selecting "File" menu > "Import..." > "General" > "Renesas GitHub FreeRTOS(with IoT libraries)
Project" will show you dialogs below.

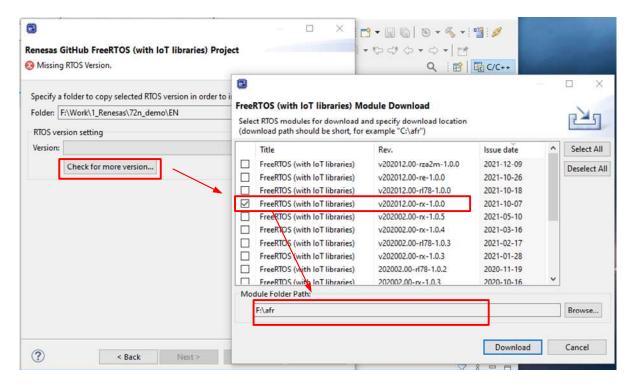


fig.3. Steps to choose FreeRTOS module to download

On a FreeRTOS (with IoT libraries) Module Download dialog, select FreeRTOS with revision "V202012.00-rx-1.0.0". Next, specify the download destination folder. Note that the folder path should be short enough to avoid errors where the path length is too long.

Once the download is complete, you will be able to see the location of your project and the version of FreeRTOS, shown in fig.4.

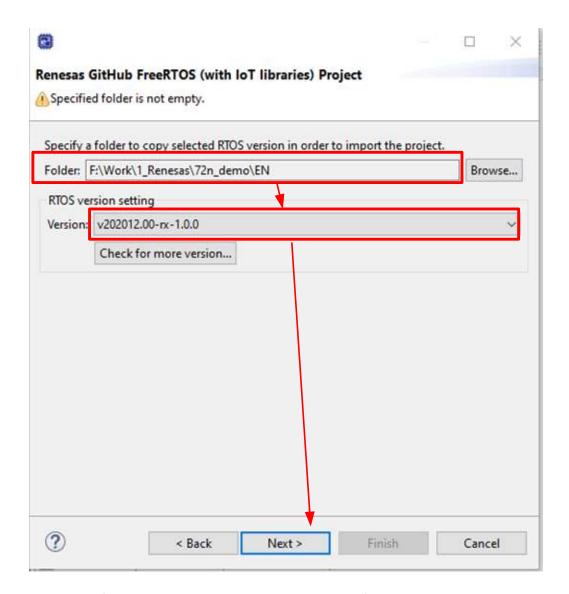


fig.4. The project location and the version of FreeRTOS to use

The folder is identical to the workspace folder and is to be referred as <base>. For the next step, you need to choose a demo type for the target MCU and the compiler.

The dialog show in fig.5 lists up products ready for import to your <base> folder. The list contains three types( aws\_demo, aws\_test and boot\_loader). Pick up

<sup>&</sup>quot;aws\_demos( ...\projects\renesas\rx72n-envision-kit\e2studio\aws\_demos)"

from the list.

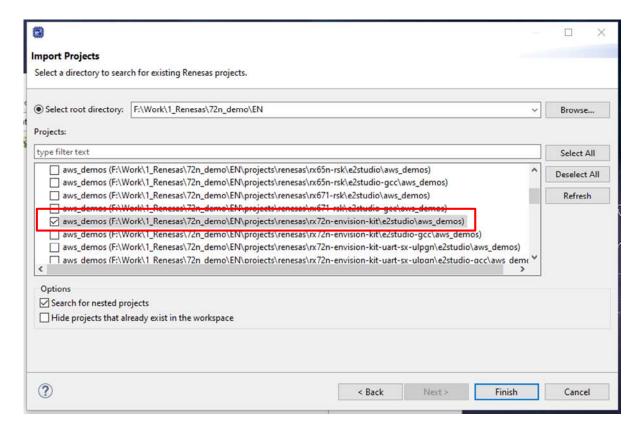
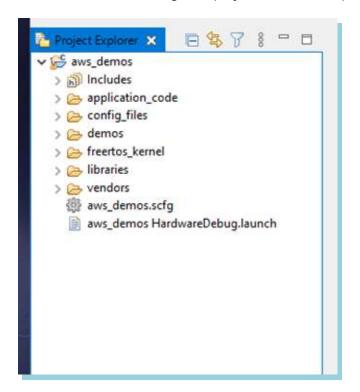


fig.5. Import Projects dialog

Script runs to extract selected source files and organize project folders in the project explore pane.



### 2.DEVICE INFO SETTINGS

Before adding FIT modules, set the board and device information. Double-click aws\_demos.scfg on the Project Explorer to open the Smart Configurator Perspective and select the Board tab at the bottom to display the "Device selection" settings pane.

In the "Board" type selection list, choose "RX72NEnvisionKit(V1.10)". If no board type listed, you can get them by clicking the link named "Download more boards...".

When you choose the board, "Device" is filled with "R5F72NDHxFB" automatically.

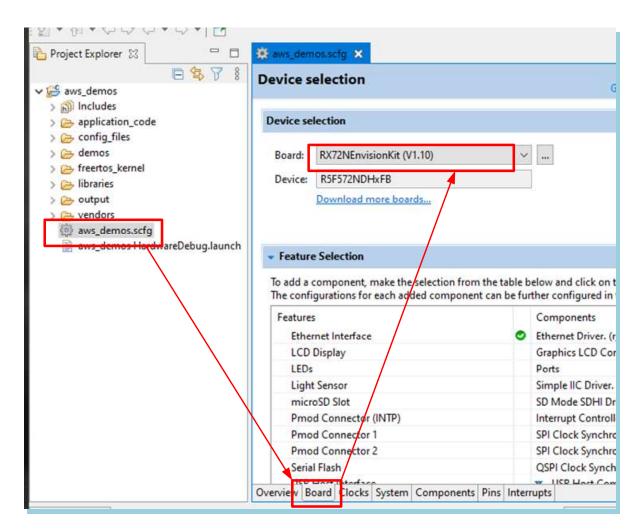


fig.7. Device selection

### 3.ADDING FIT MODULES

At this point, the project has the source files for FreeRTOS, the IoT library and the demo application. In addition, the source files of the necessary FIT components (drivers provided by

Renesas) have already been generated. However, some FIT component libraries need to be downloaded and obtained from the Renesas site.

Double-click aws\_demos.scfg on the Project Explorer to open the Smart Configurator Perspective and select the "Components" tab at the bottom to display the "Software components configuration" pane. Then push the icon to show "Software Component Selection" dialog shown in fig.8.

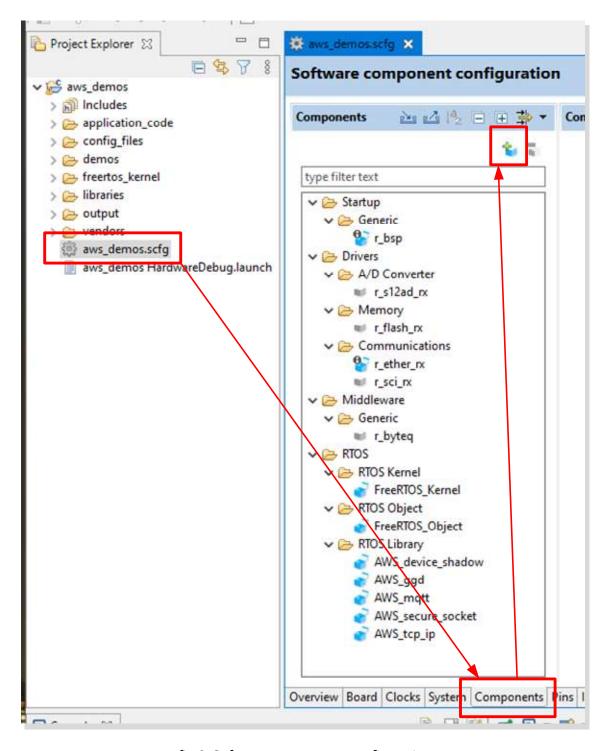


fig.8. Software components configuration

In the "Software Component Selection" dialog, find and choose one FIT component to add the project. If no components are listed in the dialog, it means that you need to download FIT components from Renesas site into your PC. Click the "Download the latest FIT drivers and middleware" link.

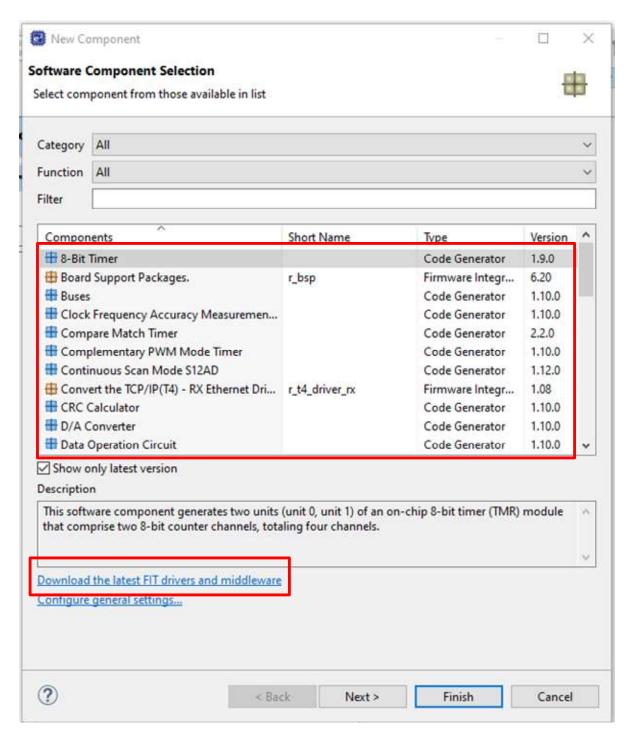


fig.9. Software Component Selection Dialog

If you have downloaded the latest FIT components in your PC, you can extract the components to add in the list by specifying the function type of the component.

wolfSSL demo requires following FIT components to add the project:

- TSIP component(r\_tsip\_rx)
- CMT component(r\_cmt\_rx)

Let's take an example of how to add a TSIP component. Select "Security" function category in the following dialog lists up TSIP in the component list. Click the TSIP in the list and then push "Finish" button to add the component. Since you can add only one FIT component at a time, repeat the same steps to add other components.

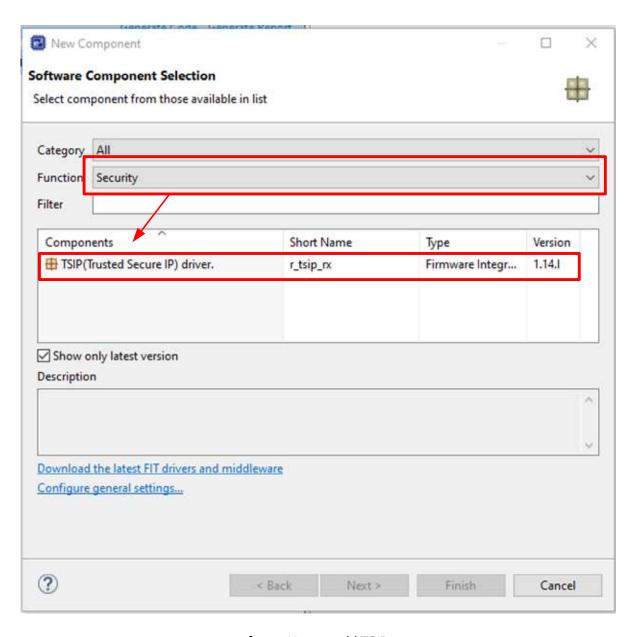


fig.10. How to add TSIP

After adding TSIP and CMT driver to the project, you can see those components are listed in the components pane.

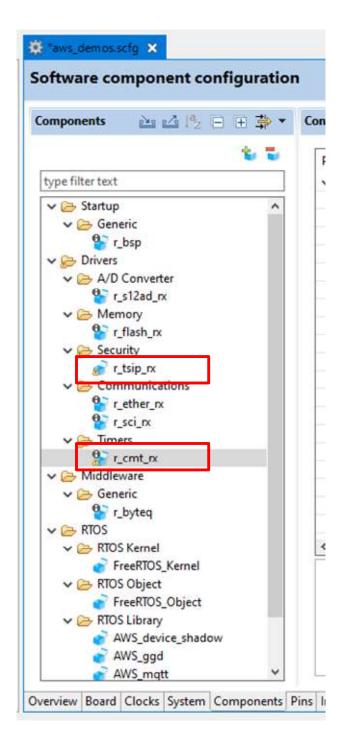


fig.11. Component list

After specifying the required FIT components, let SMC(SMart Configurator) generate source files. Push the button on the top right of the "Software component configuration" pane. Generated files are added to the aws\_demos project.

### 4. COPY WOLFSSL PACKAGE

If you have a wolfSSL package downloaded from the GitHub or wolfSSL download page, it has version string in its top folder name(such as wolfssl-5.1.1-stable) like as the right box of fig.12. Copy the entire package under the <br/>base> folder with the name "wolfssl".

This is important because both wolfSSL demo and aws\_demos refer each other by traversing their path names. Therefore, name of wolfssl top folder and the location should be exact the same as fig.12.

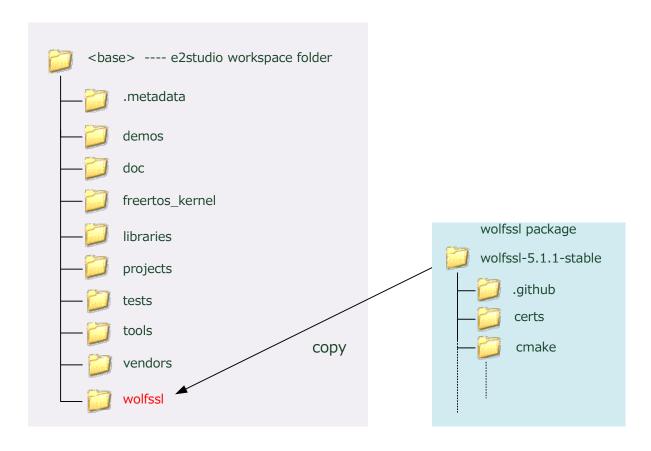


fig.12. folder structure after wolfssl package added

### **5.SECTION SETTING**

Section setting in the memory map is necessary. Open the property page of the aws\_demos project, then choose "C/C++ Build" > "Settings" > "Linker" > "Section" to show section setting pane. Push the button located in the right most of the pane to show the "Section Viewer" dialog.

Push the "Import..." button to show the dialog for specifying the section setting file to import.

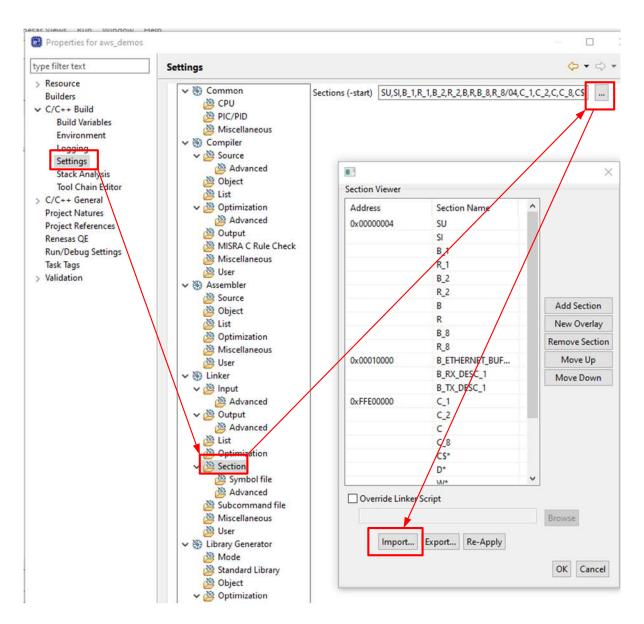


fig.13. How to show the "Section Viewer" dialog

In the dialog for a section file, specify the following file:

<base>\wolfSSL\IDE\Renesas\e2studio\RX72N\EnvisionKit\resource\section.esi

# 6.ADDING WOLFSSL LIBRARY AND WOLFSSL DEMO FILES

The next thing you need to do is add the wolfSSL library project and the code for the wolfSSL demo application to the aws\_demos project.

### IMPORTING WOLFSSL LIBRARY PROJECT

Then import the project that builds the wolfSSL library into the project explorer on e<sup>2</sup> studio. The wolfSSL library project is already available as an e<sup>2</sup> studio project in the wolfSSL package.

On e<sup>2</sup> studio, selecting "File" menu > "Open a project from the file system" > "Directory" pops up the dialog for specifying a folder including a project file. Specify the following folder:

 $< base > \ | aws\_demos \setminus IDE \setminus Renesas \setminus 2studio \setminus RX72N \setminus Envision Kit \setminus wolfssland | State | State$ 

The dialog finds out a wolfssl project to import.

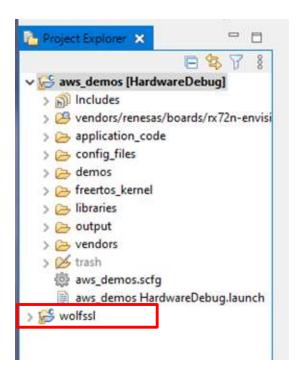


fig.14. Project structure after wolfssl project imported

You can see that the wolfssl project has been added in the project explorer. In the added wolfssl library project, there is nothing to set because the path to the include files generated by the aws\_demos project is already set.

# ADDING WOLFSSL DEMO APPLICATION FILES

Add wolfSSL demo application files which work as a kind of FreeRTOS demo task. Point to the aws\_demos folder on the project explorer pane, then open the floating dialog by right button click to create a new source file folder named "wolfSSL demo".

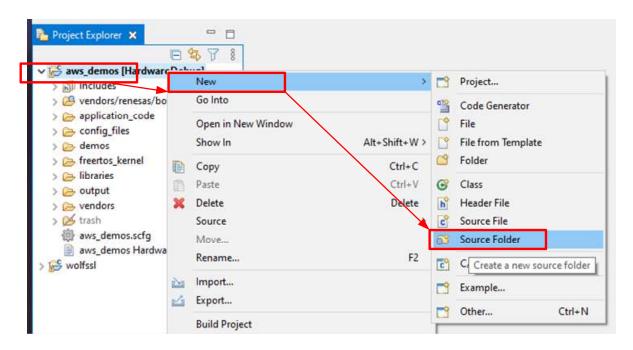
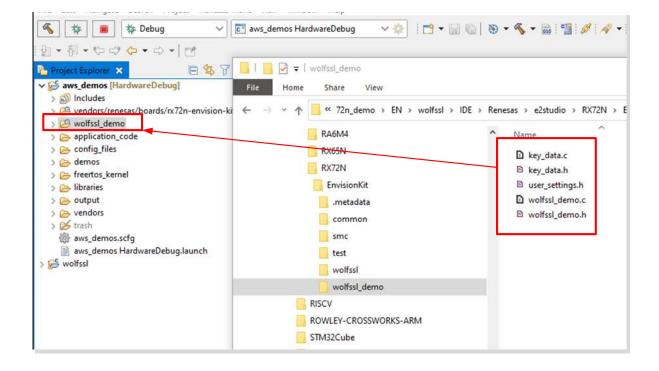


fig.15. Creating a source folder for wolfssl\_demo

Open the following folder with explorer and grab all the files (\*.c, \*.h) in there and drop them on the created "wolfssl\_demo" folder in the project explorer pane of the  $e^2$  studio.

<base>\wolfssl\IDE\Renesas\e2studio\RX72N\EnvisionKit\wolfssl\_demo



# fig.16. Adding wolfssl\_demo source files

You will be asked whether you want to process these files by copying or linking. Choose linking.

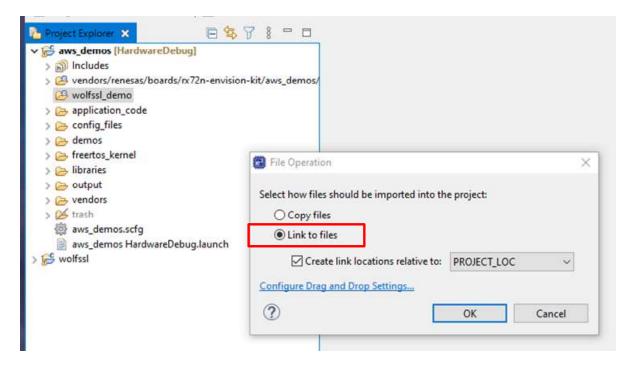


fig.17. Copying files by linking

Add more files below to the wolfssl\_demo folder by linking:

- <base>\wolfssl\wolfcrypt\benchmark\benchmark.c
- <base>\wolfssl\wolfcrypt\benchmark\benchmark.h
- 3. <base>\wolfssl\wolfcrypt\test\test.c
- 4. <base>\wolfssl\wolfcrypt\test\test.h

Finally, you should see the wolfssl\_demo folder in the project explorer, as shown in fig.18.

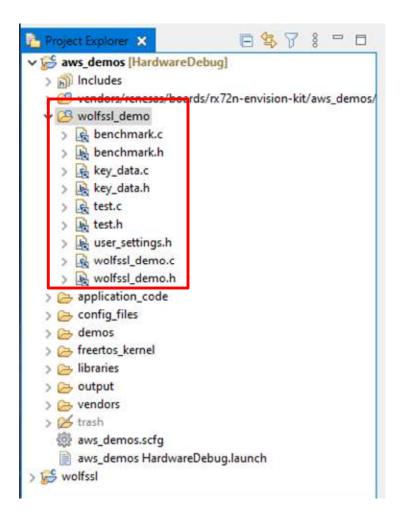


fig.18. files in the wolfssl demo folder

### ADDING INCLUDE FILE PATHS TO THE PROJECT

Open aws\_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Compiler" > "Source" to show "include file directories" pane. Add following include file paths:

- ◆ \${ProjDirPath}/../../../wolfssl
- ◆ \${ProjDirPath}/../../../wolfssl/IDE/Renesas/e2studio/RX72N/EnvisionKit/wolfssl\_demo

### ADDING PREPROCESSOR MACRO

Open aws\_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Compiler" > "Source" to show "Macro definition" pane. Add following macro definition:

◆ WOLFSSL\_USER\_SETTINGS

This macro definition lets wolfSSL demo application refer the user\_settings.h file.

### ADDING LINK LIBRARY

Open aws\_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Linker" > "Input" to show "Relocateable files, objects files and library files" pane. Add following library file:

• \${ProjDirPath}/../../../wolfssl/IDE/Renesas/e2studio/RX72N/EnvisionKit/wolfssl/Debug/wolfssl.lib

### 7.ADDING WOLFSSL DEMO AS A TASK

wolfSSL\_demo has been added as one of the demo applications to the project but not enabled yet. To do this, enable the demo and register its entry function to the demo runner environment. Open the following configuration file with editor.

<base>\venders\renesas\boards\rx72n-envision-kit\aws\_demos\config\_files\aws\_demo\_config.h

In the file, find "CONFIG\_CORE\_MQTT\_DEMO\_ENABLED" macro definition and make it commented out. Instead add definition of "CONFIG\_WOLFSSL\_DEMO\_ENABLED" macro to set wolfssl demo enable.

```
B$7 8 - 0
」プロジェクト・エクスプローラー ♡
                                                                            In av
    vendors/renesas/boards/rx72n-envision-kit/aws_demc ^
                                                                                                        #define AWS DEMO CONFIG H
  v 🕮 wolfssl demo
                                                                                                       #include "FreeRTOSApplicationConfig.h'
     > e benchmark.c
     benchmark.h
                                                                                31
                                                                                                     ⊕ /* To run a particular demo you need to define one of these.
     > key_data.c
                                                                                                           Only one demo can be configured at a time
                                                                                33
34
35
36
37
38
39
40
41
42
     > key_data.h
                                                                                                                        CONFIG CORE HTTP MUTUAL AUTH DEMO ENABLED
CONFIG CORE HTTP S3 DONNLOAD DEMO ENABLED
CONFIG CORE HTTP S3 DONNLOAD MULTITHREADED DEMO ENABLED
CONFIG CORE MOTT MUTUAL AUTH DEMO ENABLED
CONFIG CORE MOTT CONNECTION SHARING DEMO ENABLED
CONFIG DEVICE SHADOW DEMO ENABLED
CONFIG DEVICE SHADOW DEMO ENABLED
CONFIG DEVICE DEFENDER DEMO ENABLED
CONFIG DEVICE DEFENDER DEMO ENABLED
CONFIG OBSERVINGASS DISCOVERY DEMO ENABLED
     > 🔓 test.c
     > R test.h
     > user_settings.h
     > wolfssl_demo.c
     > k wolfssl_demo.h
  > 😂 application_code

✓ 

Config_files

                                                                                                       CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED
CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED
CONFIG_POSIX_DEMO_ENABLED
CONFIG_OTA_UBBASE_DAMALED
     > 🖟 aws_bufferpool_config.h
                                                                                43
44
45
46
47
48
      > aws_demo_config.h
         aws_ggd_config.h
                                                                                                                         CONFIG_OTA_UPDATE_DEMO_ENABLED
     > aws_iot_network_config.h
     aws mgtt config.h
                                                                                                        * These defines are used in iot_demo_runner.h for demo selection */
     > aws_ota_agent_config.h
                                                                                49
50
51
52
     > 🗟 aws_secure_sockets_config.h
                                                                                                           demo is configured for MOTT */
     > 🗟 aws_shadow_config.h
                                                                                                     = #if ( OTA == 0 )
                                                                                                       // #define CONFIG_CORE_MOTT_MUTUAL_AUTH_DEMO_ENABLED
#define CONFIG_WOLFSSL_DEMO_ENABLED
        aws_wifi_config.h
                                                                                53
54
55
56
57
58
        a core_http_config.h
                                                                                                    /* demo is configur
⊖ #elif ( OTA == 1 )
        a core matt config.h
     > & core_pkcs11_config.h
                                                                                                            #define CONFIG_OTA_UPDATE_DEMO_ENABLED
     > R defender_config.h
        le defender demo config.h
                                                                                                       /* Default configuration for all demos. Individual demos can override th #define democonfigDEMO_STACKSIZE ( c #define democonfigDEMO_PRIORITY ( t
        FreeRTOSApplicationConfig.h
        Recognition FreeRTOSConfig.h
```

fig.19. Add macro in the configuration file

Moreover, open the following file.

### <base>\demos\include\iot\_demo\_runner.h

Then add the two lines of code below at just before the last #else statement of the file.

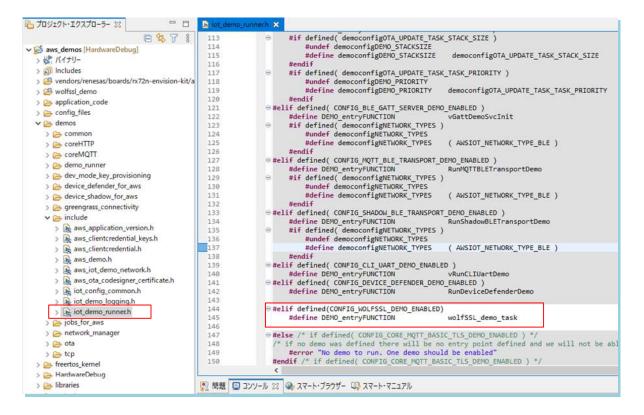


fig.20. Add definition of entry function

Those addition registers the wolfSSL\_demo to run.

# **EXECUTION OF THE WOLFSSL DEMO APPLICATION**

Internally, wolfssl\_demo has three different types of demos which is selectable by the following definitions in the wolfssl\_demo.h.

- #define CRYPT\_TST
- ◆ #define BENCHMARK
- #define TLS\_CLIENT

By enabling one of the three definition and rebuild of aws\_demo project changes the demo.

Whenever you make any change in user\_settings.h, rebuild wolfssl project followed by aws\_demos project. The execution of the demo can use debugger in the board and monitor the output from demo through "Renesas debug virtual console".

# CRYPTO-TEST DEMO

You will see the following output in the Renesas Debug Virtual Console when you choose crypto test demo.

```
🔲 Renesas Debug Virtual Console 🗶 (x)= 変数 🔍 ブレークポイント 📸 モジュール 🊹 ブロ
 wolfSSL version 5.1.1
error test passed!
MEMORY test passed!
base64 test passed!
asn
        test passed!
RANDOM test passed!
MD5 test passed!
MD4
        test passed!
SHA
        test passed!
SHA-256 test passed!
SHA-512 test passed!
Hash
      test passed!
HMAC-MD5 test passed!
HMAC-SHA test passed!
HMAC-SHA256 test passed!
HMAC-SHA512 test passed!
GMAC
       test passed!
Rabbit test passed!
       test passed!
DES
DES3
       test passed!
AES
        test passed!
AES192 test passed!
        test passed!
AES256
AES-GCM test passed!
        test passed!
PWDBASED test passed!
ECC test passed!
ECC buffer test passed!
CURVE25519 test passed!
logging test passed!
mutex
        test passed!
crypto callback test passed!
Test complete
End wolfCrypt Test
```

fig.21. Output from Crypt-test demo

# CRYPTO-BENCHMARK DEMO

You will see the following output in the Renesas Debug Virtual Console when you choose cryptobenchmark demo.

```
🔲 Renesas Debug Virtual Console 🗶 (x)= 変数 🗣 ブレークポイント 📸 モジュール 🔓 プロジェクト・エクスプローラー
                                                                                                                                      T & S & 1 7 8 8 1 16
 Start wolfCrypt Benchmark
  wolfSSL version 5.1.1
2 MB took 1.005 seconds, 2.089 MB/s
AES-128-CBC-enc 1 MB took 1.002 seconds, 1.430 MB/s
AES-128-CBC-dec 1 MB took 1.002 seconds, 1.430 MB/s
AES-192-CBC-enc 1 MB took 1.004 seconds,
AES-192-CBC-dec 1 MB took 1.010 seconds,
AES-256-CBC-enc 1 MB took 1.008 seconds,
AES-256-CBC-dec 1 MB took 1.008 seconds,
AES-256-CBC-dec 1 MB took 1.008 seconds,
                                                                                       1.313 MB/s
                                                                                       1.282 MB/s
1.211 MB/s
                                                                                        1.187 MB/s
AES-128-GCM-enc 650 KB took 1.007 seconds, 645.738 KB/s
AES-128-GCM-dec 650 KB took 1.007 seconds, 645.546 KB/s
AES-192-GCM-enc 625 KB took 1.009 seconds, 619.364 KB/s
AES-192-GCM-dec 625 KB took 1.009 seconds, 619.180 KB/s
AES-256-GCM-enc 600 KB took 1.007 seconds, 595.888 KB/s
AES-256-GCM-dec 600 KB took 1.007 seconds, 596.007 KB/s
GMAC Default 1 MB took 1.000 seconds, 1.214 MB/s
RABBIT 8 MB took 1.003 seconds, 8.253 MB/s
3DES 525 KB took 1.040 seconds, 504.856 KB/s
MD5 24 MB took 1.001 seconds, 24.397 MB/s
SHA 11 MB took 1.000 seconds, 10.984 MB/s
SHA
SHA-256
SHA-512
625 KB took 1.002 seconds, 11.651 Mb/s
SHA-512
625 KB took 1.009 seconds, 619.364 KB/s
HMAC-MD5
HMAC-SHA
11 MB took 1.001 seconds, 24.085 MB/s
HMAC-SHA
11 MB took 1.000 seconds, 10.791 MB/s
HMAC-SHA512
625 KB took 1.002 seconds, 11.428 MB/s
HMAC-SHA512
625 KB took 1.026 seconds, 609.459 KB/s
PBKDF2
672 bytes took 1.008 seconds, 666.402 bytes/s
94 ops took 1.005 sec, avg 10.691 ms,
                                          94 ops took 1.005 sec, avg 10.691 ms, 93.532 ops/sec
2 ops took 1.322 sec, avg 660.800 ms, 1.513 ops/sec
RSA
              2048 private
ECC [ SECP256R1] 256 key gen 6 ops took 1.035 sec, avg 172.467 ms, 5.798 ops/sec
                      SECP256R1] 256 agree 6 ops took 1.034 sec, avg 1/2.300 ms, 3.003 ops/sec
SECP256R1] 256 sign 6 ops took 1.044 sec, avg 174.000 ms, 5.747 ops/sec
SECP256R1] 256 verify 4 ops took 1.330 sec, avg 332.500 ms, 3.008 ops/sec
ECDHE [
 ECDSA [
ECDSA [
CURVE 25519 key gen 4 ops took 1.017 sec, avg 254.325 ms, 3.932 ops/sec
                                                    4 ops took 1.015 sec, avg 253.750 ms, 3.941 ops/sec
 CURVE 25519 agree
Benchmark complete
End wolfCrypt Benchmark
```

fig.22. output from Crypt-benchmark demo

### TLS-CLIENT DEMO

When you attempt to run TLS\_Client demo, prepare the communication opponent program(TLS server program). wolfSSL package provides TLS server example application for this purpose. The application is generated by building wolfSSL package. You can build wolfSSL on Linux(including MacOS and WSL) with gcc installed or build using Visual Studio. The following introduces the build on WSL.

# (1) When using ECDSA certificates

Since USE\_ECC\_CERT is defined in user\_settings.h on the TLS\_CLIENT side, it is set to use the Build the server-side sample program accordingly with the following configuration options. Don't forget to give "-DNO\_RSA".

```
$ cd <base>/wolfssl

$ ./autogen.sh

$ ./configure -enable-ecc -enable-dsa CFLAGS="-DWOLFSSL_STATIC_RSA -DHAVE_DSA -DHAVE_ALLCURVES -DHAVE_ECC -DNO_RSA"

$ make

$ ifconfig
```

# (2) When using RSA certificates

Also, when setting to use RSA certificates on the TLS\_CLIENT side, comment out the USE\_ECC\_CERT definition in user\_settings.h and rebuild. Correspondingly, the server-side sample builds with the following configuration options:

```
$ cd <base>/wolfssl
$ ./autogen.sh
$./configure -enable-ecc -enable-dsa CFLAGS="-DWOLFSSL_STATIC_RSA -DHAVE_DSA
-DHAVE_ALLCURVES -DHAVE_ECC"
$ make
$ ifconfig
```

If "make" command reports no error, TLS server application is ready to run. Before running the server application, get IP address of the server by typing "ifconfig". You could see IP v4 address in the console. Set the address to the TLSSERVER\_IP macro defined in wolfSSL\_demo.c.

The IP address of the target board could be set by changing value of the following macros:

- ipconfigUSE\_DHCP defined in FreeRTOSIPCOnfig.h
- configIP\_ADDRo ~ configIP\_ADDR3 defined in FreeRTOSConfig.h

For debugging purpose or when you get trouble in TCP connection, try to use static IP address for the board.

Run the TLS server application with the following command and options. The option "-v4" specifies to use the TLS1.3 protocol. If "-v3" is specified, the TLS1.2 protocol is used.

```
$ ./examples/server/server -b -v4 -i
```

The sever application waits for the client connection. Run the demo on the board to establish TLS communication with the server application. You will see the following output in the Renesas Debug Virtual Console.

TLS Client attempt to establish TLS connection six times with TLS server using six different cipher suites respectively. The cipher suites included in the output of the TLS client depends on the TLS version and the certificate type setting.

```
T & B & A F & B & B
Renesas Debug Virtual Console X
   TLS Client demo
   - TLS server address:192.168.1.14 port: 11111
    - with TSIP
cipher : ECDHE-RSA-AES128-GCM-SHA256
Received: I hear you fa shizzle!
cipher: ECDHE-RSA-AES128-SHA256
Received: I hear you fa shizzle!
cipher: AES128-SHA
Received: I hear you fa shizzle!
cipher: AES128-SHA256
Received: I hear you fa shizzle!
cipher: AES256-SHA
Received: I hear you fa shizzle!
cipher: AES256-SHA256
Received: I hear you fa shizzle!
End of TLS_Client demo.
```

fig.23. output from TLS-Client demo

### THINGS TODO WHEN USING USER'S ROOT CA CERT

The Root CA certificate, the server certificate and the client certificate used in this example application, can be used only for evaluation. If you want to use your own certificate, prepare following items:

- 1. Provisioning Key
- 2. RSA key pair for validating RootCA certificate
- 3. The signature generated by the RootCA certificate with the private ky in 2 above.
- 4. Private key corresponding to the public key included in the client certificate.

Refer to the manual provided by Renesas for how to generate them.

# **LIMITATIONS**

WolfSSL, which supports TSIPv1.15, has the following functional restrictions.

- Message packets exchanged with the server during the TLS handshake are stored in plaintext in memory. This is used to calculate the hash of handshake messages. The content will be deleted at the end of the session.
- 2. TLS1.2 does not support the client authentication function using TSIP. Use wolfSSL\_CTX\_use\_certificate\_buffer or wolfSSL\_CTX\_use\_certificate\_chain\_buffer\_format to load the client certificate and wolfSSL\_CTX\_use\_PrivateKey\_buffer to load the private key. It is processed by software.
- 3. In TLS 1.3, the client authentication function using TSIP is supported only for ECDSA client certificates. In the case of RSA certificate, it will be processed by software.
- 4. In TLS1.3, among the server authentication functions using TSIP, "CertificateVerify" message is processed by software.
- 5. Session resumption and early data using TSIP are not supported.

The above restrictions 1 through 4 are expected to be improved by TSIP from the next version onwards.

# **RESOURCES**

Followings are the links to the sites that contain helpful information regarding board, MCU, TSIP and wolfSSL .

### **RENESAS SITES**

- Renesas wiki page for RX72N Envision Kit (<a href="https://github.com/renesas/rx72n-envision-kit/wiki">https://github.com/renesas/rx72n-envision-kit/wiki</a>)
- Renesas RX MCUs( <a href="https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus/">https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus/</a>
- Renesas Trusted Secure IP Driver(TSIP) (<u>Renesas Trusted Secure IP</u> <u>Driver(TSIP)(https://www.renesas.com/us/en/software-tool/trusted-secure-ip-driver/)</u>

# **WOLFSSL SITES**

- wolfSSL Website (www.wolfssl.com)
- wolfSSL Renesas page (<a href="https://www.wolfssl.com/docs/renesas/">https://www.wolfssl.com/docs/renesas/</a>)
- wolfSSL TSIP support page (<a href="https://www.wolfssl.com/docs/wolfssl-renesas-tsip/">https://www.wolfssl.com/docs/wolfssl-renesas-tsip/</a>)
- wolfSSL Renesas GitHub repo (<a href="https://qithub.com/wolfSSL/Renesas/">https://qithub.com/wolfSSL/Renesas/</a>)

# SUPPORT AND CONTACT

For support inquiries and questions, please email <a href="mailto:support@wolfssl.com">support@wolfssl.com</a>. Feel free to reach out to <a href="mailto:info@wolfssl.jp">info@wolfssl.jp</a>.