TCP Client Socket over SSL Application

User guide

Version 0.2

May 2016

Redpine Signals, Inc.
2107 N. First Street, #540
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com

About this Document

This document describes the process of bringing up the RS9113 based module as a SSL client in station mode.

Disclaimer:

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only. Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.

Table of Contents

Table of Figures

Table of Tables

No table of figures entries found.

1 Introduction

This project is applicable to all the WiSeConnect variants like WiSeConnect Plus, WiSeMCU and WyzBee. The term WiSeConnect refers to its appropriate variant.

1.1 SSL Overview

SSL stands for Secure Sockets Layer. SSL is the standard security technology for establishing an Encrypted link between a web server and a browser. This link ensures that all data passed between the web servers and the browsers remain Private & Integral.

Data encryption, Server authentication, Message integrity, Optional client authentication for a TCP/IP connection are the main objectives of SSL protocol**.**

1.2 Application Overview

1.2.1 Overview

This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data on socket.

1.2.2 Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open TCP Server socket over SSL at Access point using openssl.
- Establish TCP connection over SSL with TCP server opened on remote peert.
- Send TCP data from WiSeConnect device to remote device

1.3 Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

1.3.1 SPI based Setup Requirements

- Windows PC1 with CooCox IDE
- Spansion (MB9BF568NBGL) micro controller

Note: If user does not have Spansion (MB9BF568NBGL) host platform, please go through the SPI-Porting guide **\sapis\docs\RS9113-WiSeConnect-SAPI-Porting-Guide-vx.x.pdf** for SAPIs porting to that particular platform.

- WiSeConnect device
- WiFi Access point
- Windows PC2
- TCP server over SSL running in Windows PC2 (This application uses OpenSSL to create TCP server over SSL)

Note: Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

1.3.2 UART/USB-CDC based Setup Requirements

- Windows PC with Dev-C++ IDE
- WiSeConnect device
- WiFi Access point
- Windows PC2
- TCP server over SSL running in Windows PC2 (This application uses OpenSSL to create TCP server over SSL)

Note: Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

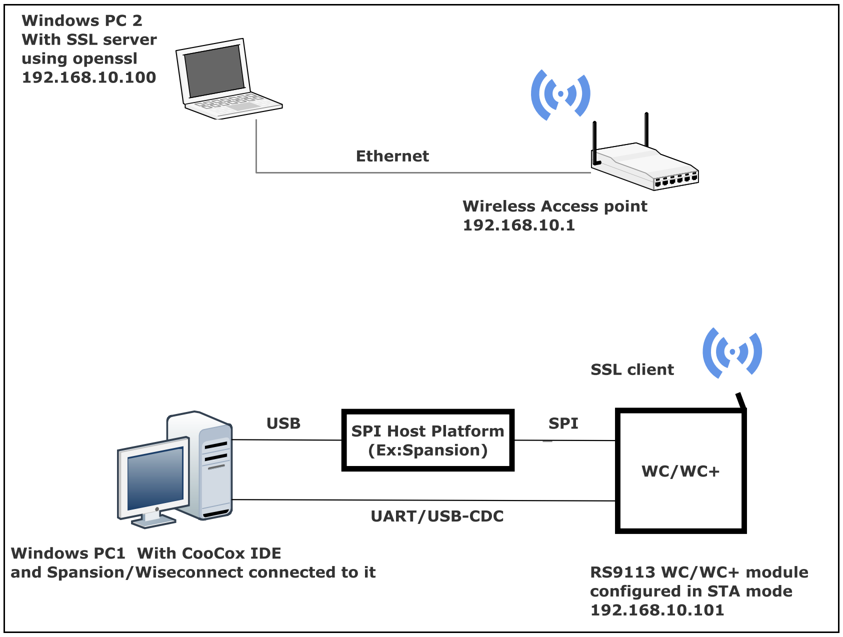


Figure 1: Setup Diagram

2 Configuration and Execution of the Application

The example application is available in the Release at **{Release $}/host/sapis/examples.** These examples will have to be initialized, configured and executed to test the application. The initialization varies based on the interface but configuration and execution are the common.

2.1 Initializing the Application

2.1.1 SPI Interface

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for opening the ***ssl_client*** example in CooCox IDE.

2.1.2 UART/USB-CDC Interface

If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for opening the ***ssl_client*** example in Dev-C++ IDE

2.2 Configuring the Application

1. Open ***sapis/examples/wlan/ssl_client/rsi_ssl_client.c*** file and update/modify following macros:

SSID refers to the name of the Access point.

```
#define SSID                         "<ap name>"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO           0
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

Valid configuration is:

RSI_OPEN - For OPEN security mode

RSI_WPA - For WPA security mode

RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE            RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                          "<psk>"
```

To Load certificate

```
#define  LOAD_CERTIFICATE          1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.

By default, application loading “cacert.pem” certificate if **LOAD_CERTIFICATE** enable. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package “***sapis/examples/utilities/certificates/certificate_script.py***”

Ex: If the certificate is wifi-user.pem .Give the command in the following way

```
python certificate_script.py ca-cert.pem
```

Script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

- After conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.

Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

So define LOAD_CERTIFICATE as 0, if certificate is already present In the Device.

Note: All the certificates are given in the release package.
Path: `sapis/examples/utilities/certificates`

Enable **SSL** macro to open SSL socket over TCP.

```
      #define SSL                    1
```

DEVICE_PORT port refers SSL client port number

```
      #define DEVICE_PORT                5001
```

SERVER_PORT port refers remote SSL server port number which is opened in Windows PC2

```
      #define SERVER_PORT                5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with SSL server socket.

IP address should be in long format and in little endian byte order.

Example: To configure “192.168.0.100” as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
      #define SERVER_IP_ADDRESS      0x6400A8C0
```

NUMEBR_OF_PACKETS refers how many packets to send from TCP client

```
      #define NUMBER_OF_PACKETS     <no of packets>
```

Application memory length which is required by the driver

```
      #define  GLOBAL_BUFF_LEN       8000
```

To configure IP address

DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
      #define DHCP_MODE   1
```

Note: If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

(Or)

If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
      #define  DEVICE_IP          0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
      #define  GATEWAY            0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
      #define   NETMASK           0x00FFFFFF
```

2. Open ***sapis/include/rsi_wlan_config.h*** file and update/modify following macros :

```
#define  CONCURRENT_MODE              RSI_DISABLE
#define  RSI_FEATURE_BIT_MAP          FEAT_SECURITY_OPEN
#define  RSI_TCP_IP_BYPASS            RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_CLIENT
                                  | TCP_IP_FEAT_SSL)
 #define  RSI_CUSTOM_FEATURE_BIT_MAP  0
 #define  RSI_BAND                    RSI_BAND_2P4GHZ
```

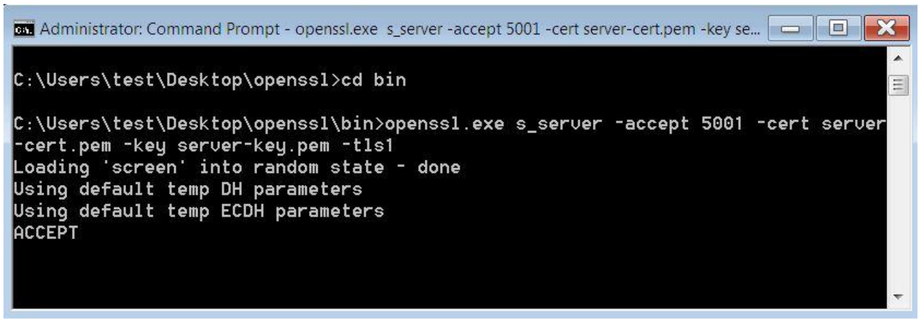
2.3 Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect WiSeConnect device in STA mode.
2. In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command,

```
Openssl.exe s_server  -accept<SERVER_PORT> -cert
<server_certificate_file_path> -key <server_key_file_path> -
tls<tls_version>
```

Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1.

Note: All the certificates are given in the release package.
Path: `sapis/examples/utilities/certificates`



Administrator: Command Prompt - openssl.exe s_server -accept 5001 -cert server-cert.pem -key se...

```
C:\Users\test\Desktop\openssl>cd bin

C:\Users\test\Desktop\openssl\bin>openssl.exe s_server -accept 5001 -cert server
-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```

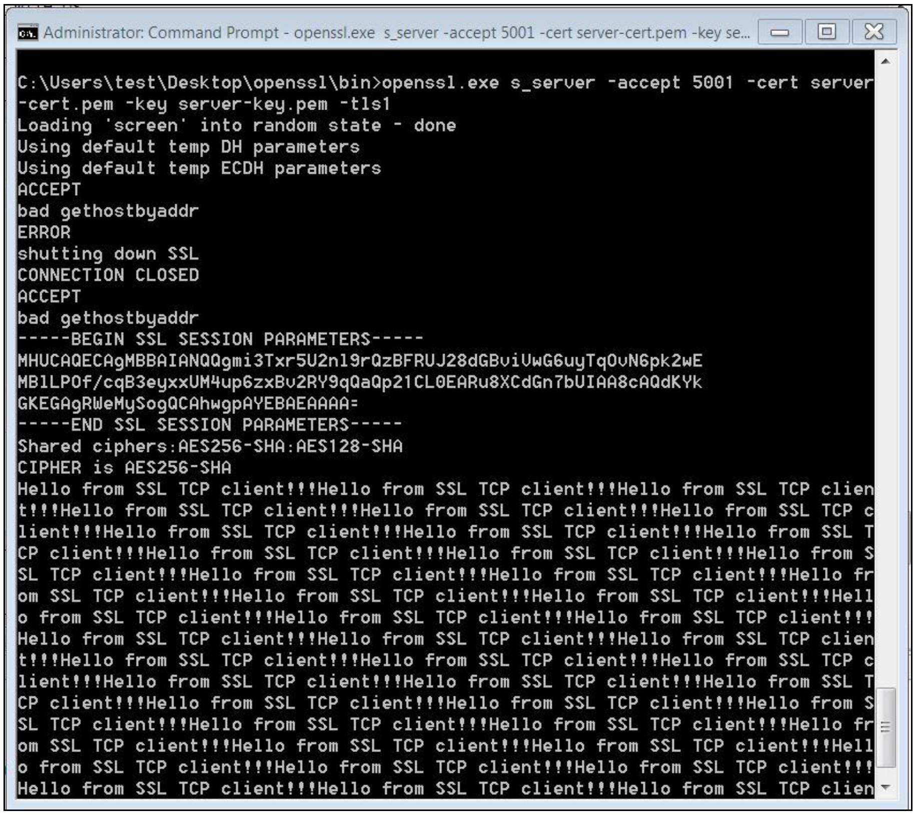
3. **SPI Interface**

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for executing the ***ssl_client*** example in CooCox IDE.

4. **UART/USB-CDC Interface**

If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for executing the ***ssl_client*** example in Dev-C++ IDE

5. After the program gets executed, WiSeConnect Device would be connected to Access point having the configuration same that of in the application and get IP.
6. The Device which is configured as SSL client will connect to remote SSL server and sends number of packets configured in **`NUMBER_OF_PACKETS`**


Administrator: Command Prompt - openssl.exe s_server -accept 5001 -cert server-cert.pem -key se...

```
C:\Users\test\Desktop\openssl\bin>openssl.exe s_server -accept 5001 -cert server
-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
bad gethostbyaddr
ERROR
shutting down SSL
CONNECTION CLOSED
ACCEPT
bad gethostbyaddr
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMBBAIANQQgmi3Txr5U2nl9rQzBFRUJ28dGBviVwG6uyTqOvN6pk2wE
MBlLPOf/cqB3eyxxUM4up6zxBv2RY9qQaQp21CL0EARu8XCdGn7bUIAA8cAQdKYk
GKEGAgRWeMySogQCAhwgpAYEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:AES256-SHA:AES128-SHA
CIPHER is AES256-SHA
Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP clien
t!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP c
lient!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL T
CP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from S
SL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello fr
om SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hell
o from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!
Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP clien
t!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP c
lient!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL T
CP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from S
SL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello fr
om SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hell
o from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP client!!!
Hello from SSL TCP client!!!Hello from SSL TCP client!!!Hello from SSL TCP clien
```