Secured TCP Client using MPLAB Harmony 3

**Overview**

This training material is intended to demonstrate **how to modify existing TCP client for SSL/TLS secure messaging using Harmony V3**. It is not intended to show the details about creating a simple and unencrypted TCPIP client application. For more details about creating a simple TCPIP client, please visit the link below. This training material do not contain getting started on MPLAB Harmony V3.

This training material discussed the basics of cryptography and other relevant topics related to SSL/TLS.

**MPLAB Harmony 3 Installation**

<https://microchipdeveloper.com/harmony3:mhc-overview>

**MPLAB harmony 3 Package Documentation on GitHub**

<https://github.com/Microchip-MPLAB-Harmony>

**MPLAB Harmony 3 NET Support Package**

<https://github.com/Microchip-MPLAB-Harmony/net>

**Creating your first TCP/IP Application**

<https://github.com/Microchip-MPLAB-Harmony/net/wiki/Create-your-first-tcpip-application>

**WolfSSL Manual**

<https://www.wolfssl.com/docs/wolfssl-manual/>

Training Plan

**Unsecured to Secured TCP Client using MPLAB Harmony 3**

[Pre-training 3](#_Toc69759666)

[Cryptography 3](#_Toc69759667)

[Types of Cryptography 3](#_Toc69759668)

[Secure Socket Layer 4](#_Toc69759669)

[Certificates 5](#_Toc69759670)

[Self-signed Certificate 6](#_Toc69759671)

[Creating Your First Secured TCPIP Client 7](#_Toc69759672)

[Setup 7](#_Toc69759673)

[Harmony 3 Framework Configuration 8](#_Toc69759674)

[Configuring WolfCrypt and WolfSSL 12](#_Toc69759675)

[Application Programming Interface 14](#_Toc69759676)

[Example Code 16](#_Toc69759677)

[Application Commands 16](#_Toc69759678)

[Sequence Diagram 17](#_Toc69759679)

[Running the example code 17](#_Toc69759680)

[Debugging 17](#_Toc69759681)

[Learning Challenge 18](#_Toc69759682)

[References 19](#_Toc69759683)

# Pre-training

## Cryptography

Cryptography is used to encrypt/decrypt message for security. The word came from *“Kryptós”* meaning hidden or secret and *“Graphein”* meaning writing or to write.

In modern security, the focus is on creating codes/algorithm that are computationally impossible to break or could take years to break.

### Types of Cryptography

**Symmetric Cryptography**

This type of cryptography uses a unique key called *secret key* and is located on both server and client.

Below is how symmetric cryptography is used for secure communication:

1. When the client has an unencrypted message, the client will use its own secret key to encrypt the message using an encryption algorithm.
2. The client will send the encrypted message.
3. Then, the server will receive the encrypted message.
4. Since the server has the same secret key of the client, it can decrypt the client encrypted message using his own secret key.

The advantage of using symmetric cryptography is the keys are much shorter and actually, only one generated key is needed and then, is duplicated for the other party. Both can initiate communication from another since they basically have the same keys.

If the secret key is duplicated by a third party, the encrypted message can be easily decrypted and the third party can initiate communication on both server and client. These are the disadvantages of using symmetric cryptography.

**Asymmetric Cryptography**

Asymmetric cryptography, also known as public key cryptography, uses two unique set of keys called private and public keys. Usually, private keys are located to servers while public keys are located to the clients.

Here’s how asymmetric cryptography was used for security:

1. Initially, the server has to generate a private key.
2. The server will generate the public key using its private key by an encryption algorithm.
3. Then, the server will provide the public key to the clients.
4. Client messages can be encrypted using the public keys.

In asymmetric cryptography, the only server has the capability to decrypt client encrypted message. This type of cryptography is more secure because of this advantage.

Public keys use longer key length which makes it slower to generate.

## Secure Socket Layer

Secure socket layer or SSL and its successor, Transport Layer Security or TLS, are protocols that uses cryptography to provide secure communication on a network. Applications are usually on top of SSL/TLS to provide security from its end application.

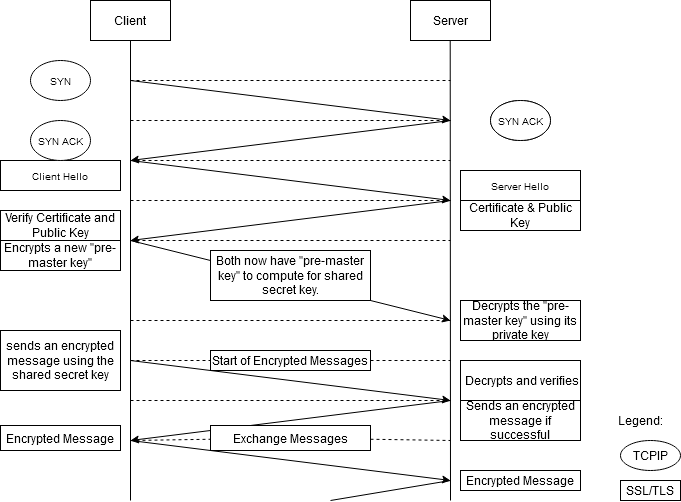


Figure SSL/TLS Handshake

Here’s a quick explanation of the SSL/TLS handshake:

1. The communication usually starts in unencrypted communication.
2. The client will send cryptographic info, like its supported ciphers (encryption algorithm) and its SSL/TLS version, to the server. This is called ***Client Hello****.*
3. Then, the server will negotiate ciphers based from the client supported ciphers. If the server does not support the client ciphers, it will drop the connection. The server also sends its server certificate and the *public key*. This is called ***Server Hello***.
4. The client checks the certificate. If it checks out, the client will compute for the pre-master key and then encrypt it with the public key. The client will send this to the server.
5. The server will decrypt this message to get the pre-master key using its own private key.

At this point, both now have a pre-master keys and they will generate the shared secret/master/session key.

1. Client sends an encrypted message using the shared secret key with changed cipher specs.
2. The server will decrypt the message using the shared secret key and will send an encrypted message with changed cipher specs.
3. If both succeeded to decrypt the message, they will start exchange of encrypted messages.

In summary, initiating communication starts in unencrypted protocol. Then, asymmetric encryption/cryptography is used at the beginning of the handshake. They will use this type of encryption to securely transport shared/secret keys (symmetric cryptography) to start encrypted exchanging messages moving forward.

## Certificates

A certificate binds a public key to an identity like website or domain. Actually, they contain the physical identity of a server such as the example below. They are commonly x.509 format which are usually found on our browsers.

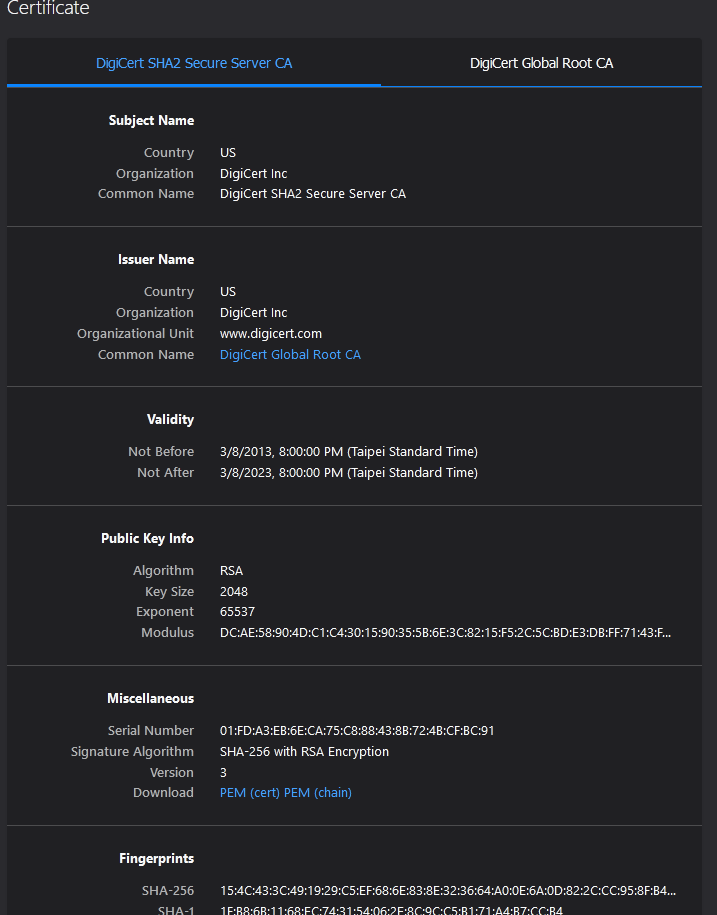


Figure Example of X.509 Formatted Certificate

Certificates are part of Public Key Infrastructure or PKI. This infrastructure has the following roles:

* Manages and distributes public key to the clients.
* Issuing parties are called ***Certificate Authority***.
* Ensures certificates are authentic.

Certificates are provided by the Certificate Authority. For the server to have its own certificate from the certificate authority, it needs to generate ***Certificate Signing Request*** or CSR.

We will not get into the details of Public Key Infrastructures.

For this training, we will not be using CSR. Instead, we are root certificate authority (so no need for CSR). We should generate private key as the certificate authority and we will use this private key to sign the server certificate. This is called ***self-signed certificate***.

### Self-signed Certificate

First, please ensure that “openssl” is installed on the machine. If openSSL is installed, its version should appear when “openssl version” is typed in the terminal.

1. On the server, generate a private key using the code below:

$openssl genrsa 1024 > testserver -key.pem

* 1. The command above will generate an RSA private key.

1. Generate a server certificate using the private key we had created above using the code below:

$o*penssl req -new -x509 -nodes -sha256 -days 1000 -key* testserver\_private-key.pem *>* testserver-cert*.pem*

1. Fill-up the information that will appear on the certificate.
2. Now, we have created a self-signed certificate. The generated private key (*testserver -key.pem*) was used to sign the generated certificate (*testserver-cert*.pem).
3. We should add the certificate and our private key to our server.
4. On the client, we only need the certificate. Harmony V3 accepts DER formatted (binary) certificate so we may need to convert our PEM certificate to DER format using the code below:

$openssl x509 -in testserver-cert.pem -inform PEM -out testserver-cert.der -outform DER

1. This DER format (binary) certificate should be converted to hex (array). For simplicity of this training, we will be using an online tool, <https://www.browserling.com/tools/file-to-base64>, to instantly convert this file to hex (array). Note: Use C/C++ conversion.



Figure Bin2Hex Online Tool

1. Then, we should create our own header file, my-test-server-certificate.h, for the byte array. Please see the file “my-test-server-certificate.h”.

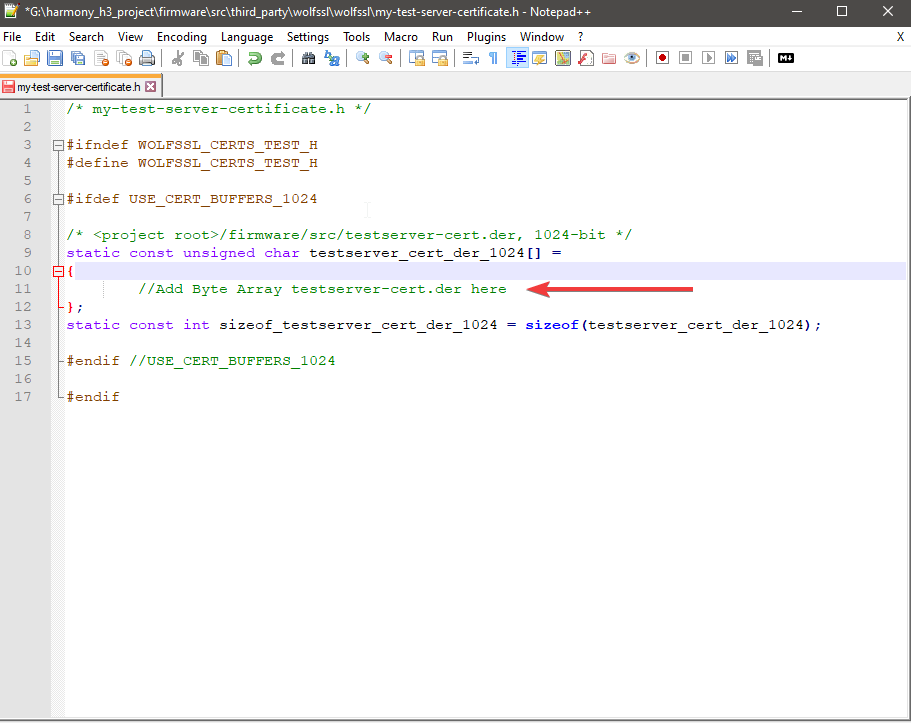


Figure Template for Certificate Header File

Please store this header file in <Harmony v3 Project>\firmware\src\third\_party \wolfssl\wolfssl. We will use this header file later on in our Harmony V3 project.

# Creating Your First Secured TCPIP Client

## Setup

1. PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit (Crypto) DM320008-C (PIC32MZ2064DAB288) with LAN8740

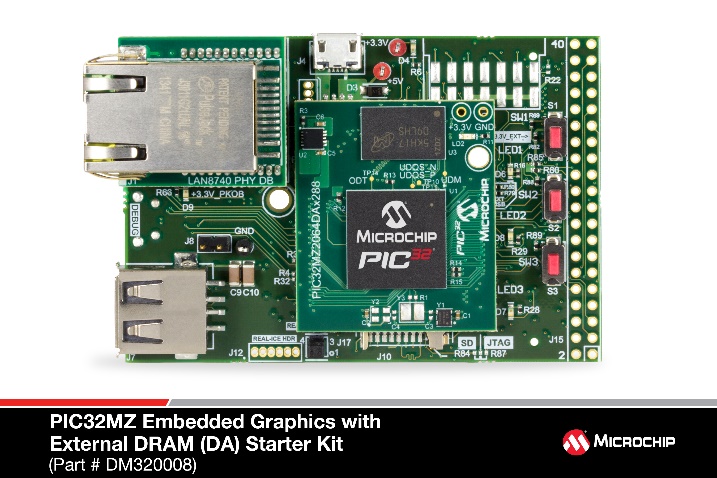


Figure DM320008

* Make sure heap size is >300 kB and leave Command Print Buffer Size as is.
* Ensure that UART pins are configured as follows:
  + RG9 – U2TX
  + RB0 – U2RX
* The ethernet pins should also be configured properly. For this board,
  + RD6 – ETXEN
  + RJ8 – ETXD0
  + RJ9 – ETXD1
  + RH8 – ERXD0
  + RH5 – ERXD1
  + RF3 – ERXERR
  + RH13 – ERXDV
  + RJ11 – ERXCLK
  + RD11 – EMDC
  + RJ1 – EMDIO
  + RJ15 – GPIO (nRST pin of LAN8740 and should be configured as OUT/HIGH)

1. WolfSSL Server Example in Linux
   * Taken from WolfSSL Examples Github in testing the client. Please see reference section for the link.

## Harmony 3 Framework Configuration

1. The initial step in setting up Harmony V3 to use SSL/TLS is to follow the guide below: <https://github.com/Microchip-MPLAB-Harmony/net/wiki/Create-your-first-tcpip-application>.
2. After checking if the setup above works, we can continue adding libraries/modules that are used for SSL/TLS.
3. Add **SNTP** under Application Layer. This is a dependency of WolfCrypt.

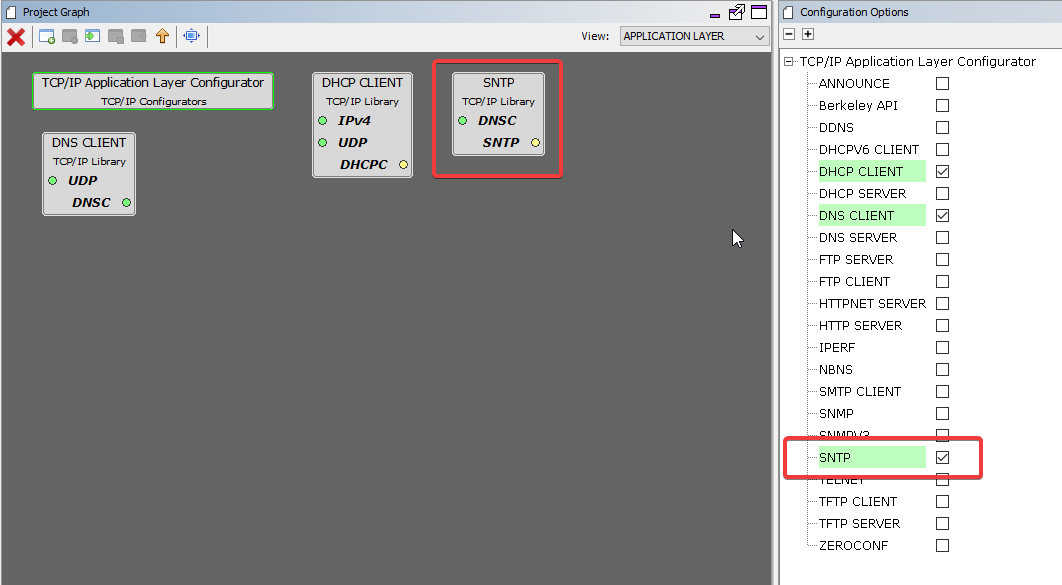


Figure Adding SNTP

1. We can add **Network Presentation Layer 6** from Libraries > TCPIP > Layer6-Presentation.

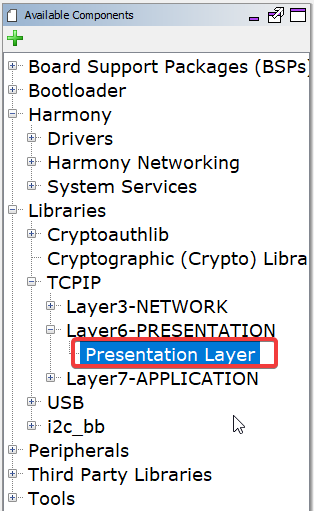


Figure Adding Presentation Layer From Available Components

1. The following view should appear. If not, you can navigate to the Presentation Layer under Views.

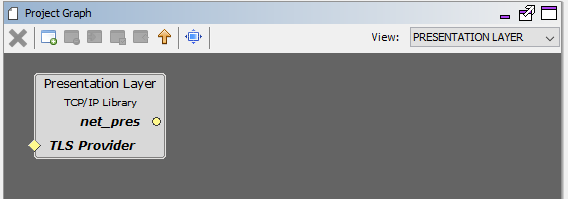


Figure Presentation Layer Added to Project Graph

1. Then, go to **Transport Layer**. We should add ***WolfCrypt***.
   1. Uncheck “**Disable Crypto Dependency**” then click “Yes” when activating “Crypto Library”.

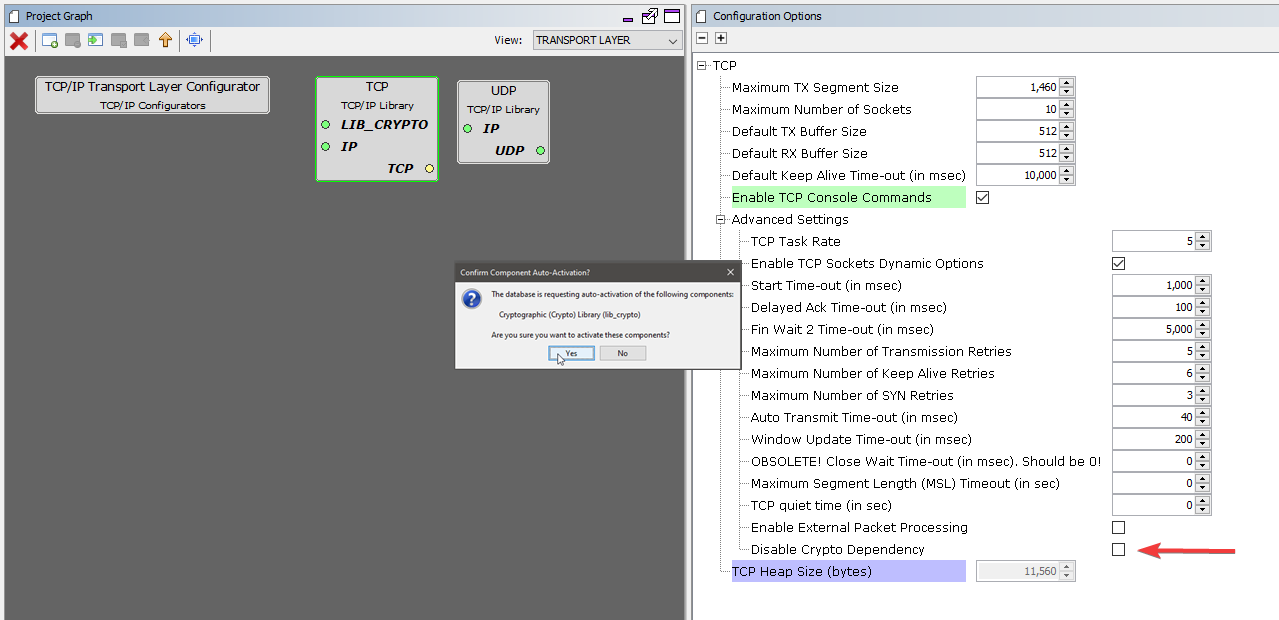


Figure Enable Crypto Dependency and Activating Crypto Library

* 1. After that, add **WolfCrypt** Library on the next prompt.

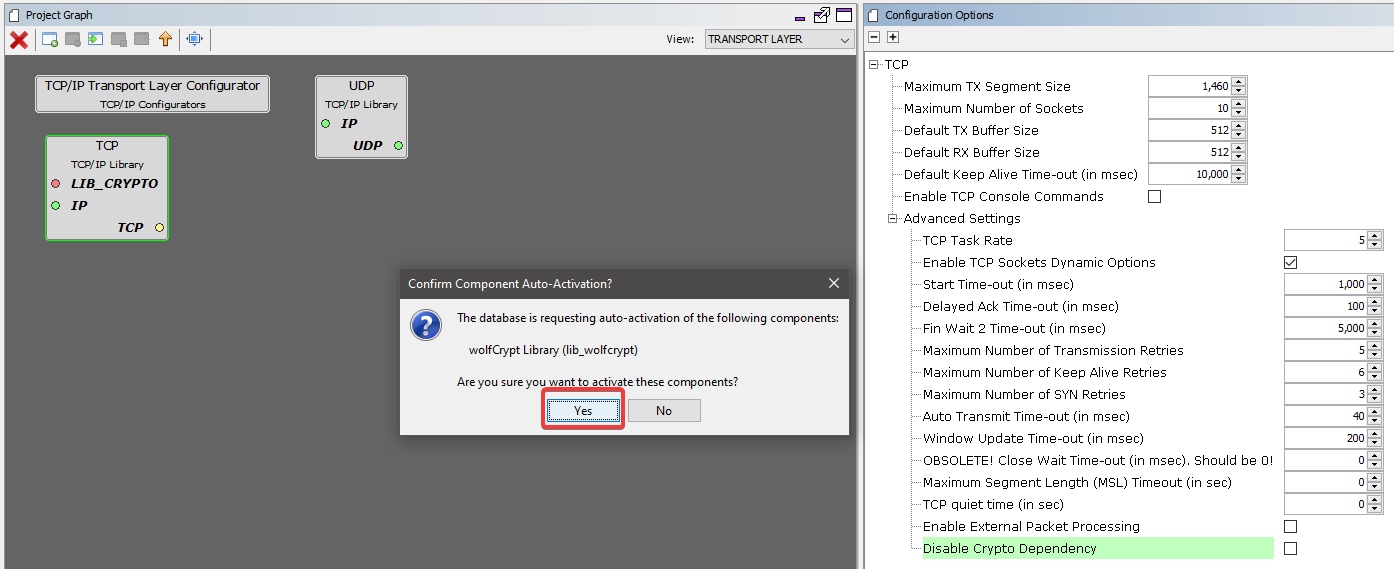


Figure Adding WolfCrypt

1. Back to **root view**, we should add **WolfSSL** Library as a consumer of Wolf Crypt.

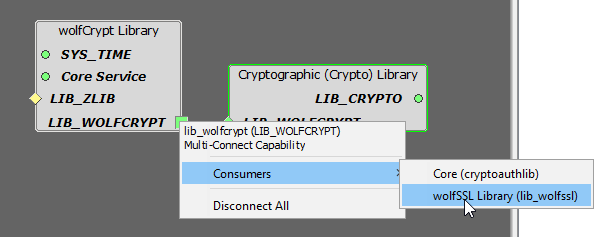


Figure Adding WolfSSL Library

1. After that, we should add **WolfSSL** in TCPIP Stack as its TLS provider.

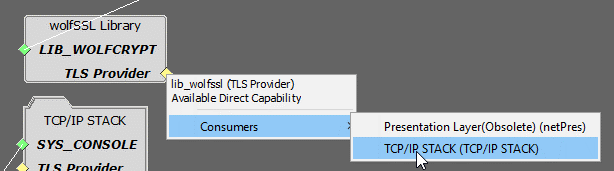


Figure Adding WolfSSL Library as the TLS Provider of TCPIP Stack

1. The project graph root view should appear similar as shown below:

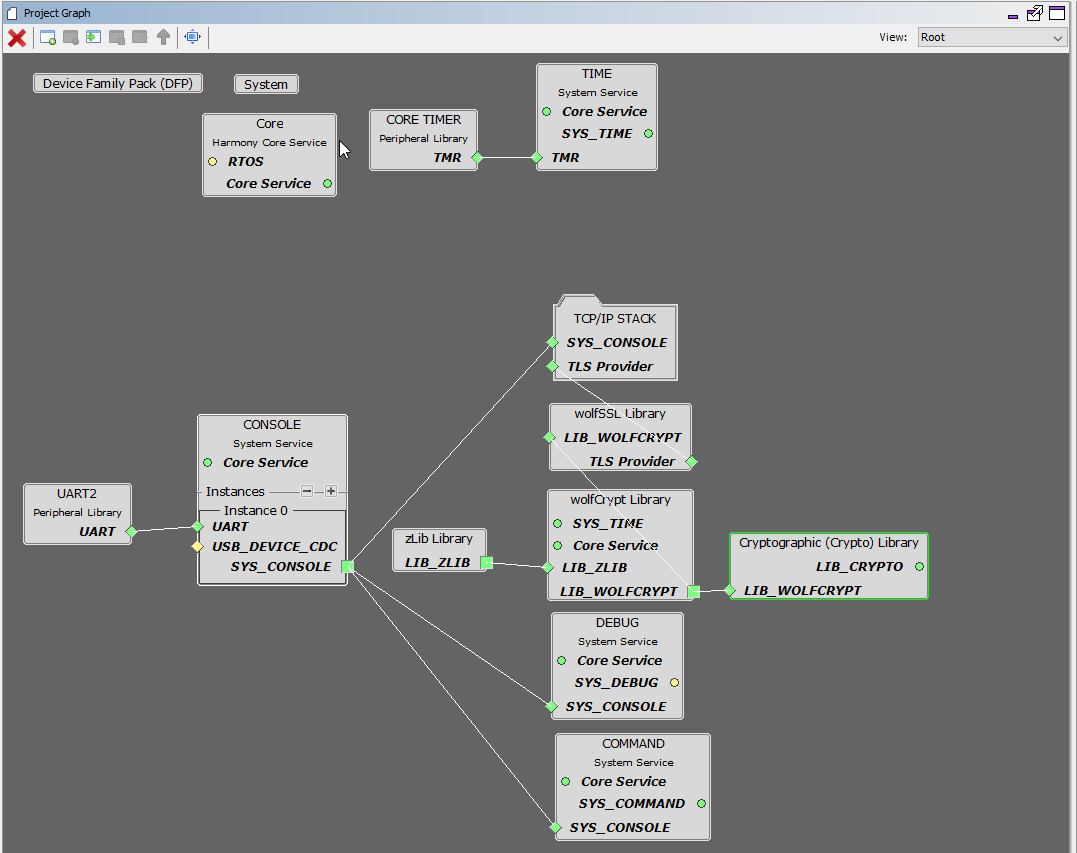


Figure Project Graph Root View

## Configuring WolfCrypt and WolfSSL

1. The client configuration below is based on a WolfSSL server example that supports the following configuration:
   1. Cipher Suite: **TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384**
   2. Self-Signed Certificate (DER format)
      1. This is what we have generated in self-signed certificate section.
2. In WolfCrypt Library, please ensure the following are checked and leave others as is:
   1. The settings below should allow our client to support the cipher suite above.

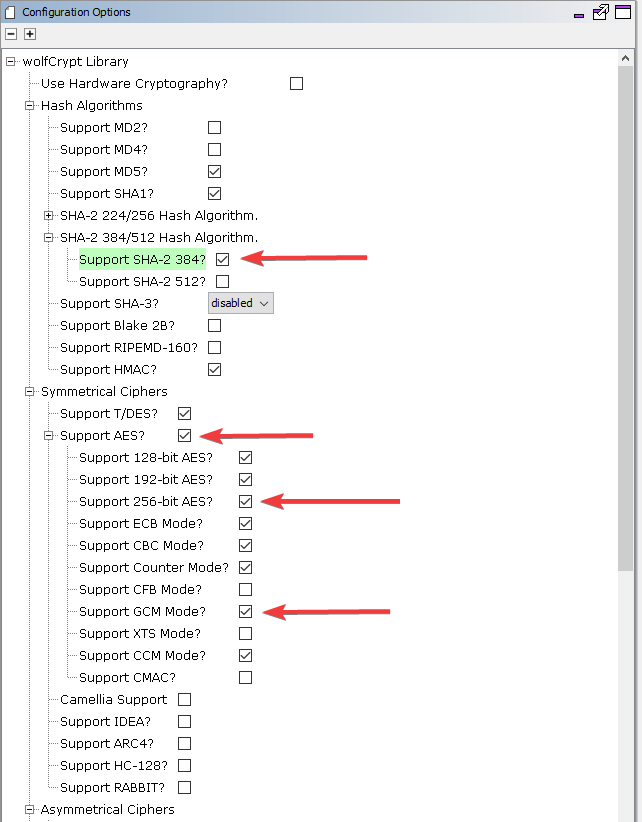


Figure WolfCrypt Library Configuration 1

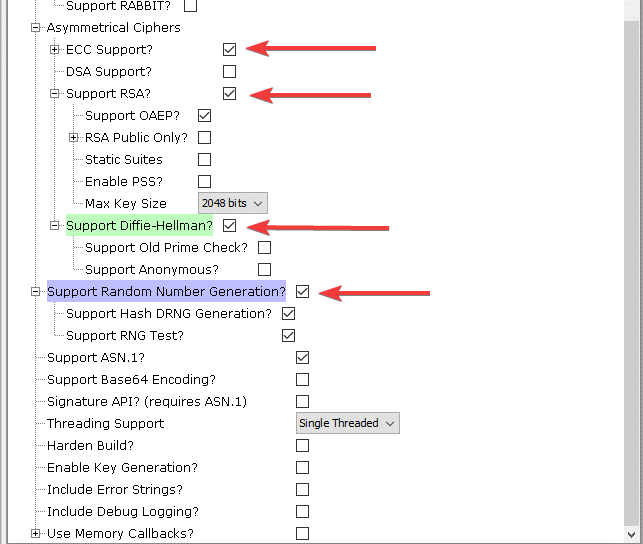


Figure WolfCrypt Library Configuration 2

1. In WolfSSL Library, please ensure that the following are checked and leave others as is:
   1. Enable Debug Messaging to enable WolfSSL console output.

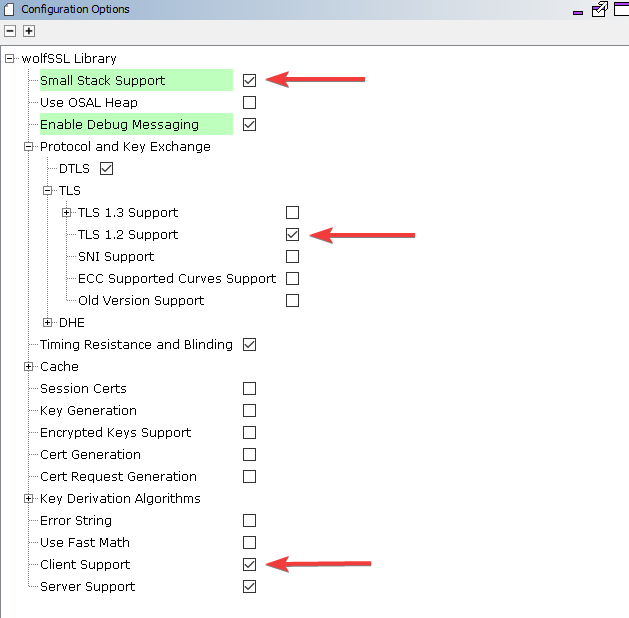


Figure WolfSSL Library Configuration

1. We should also configure the presentation layer with the settings below:

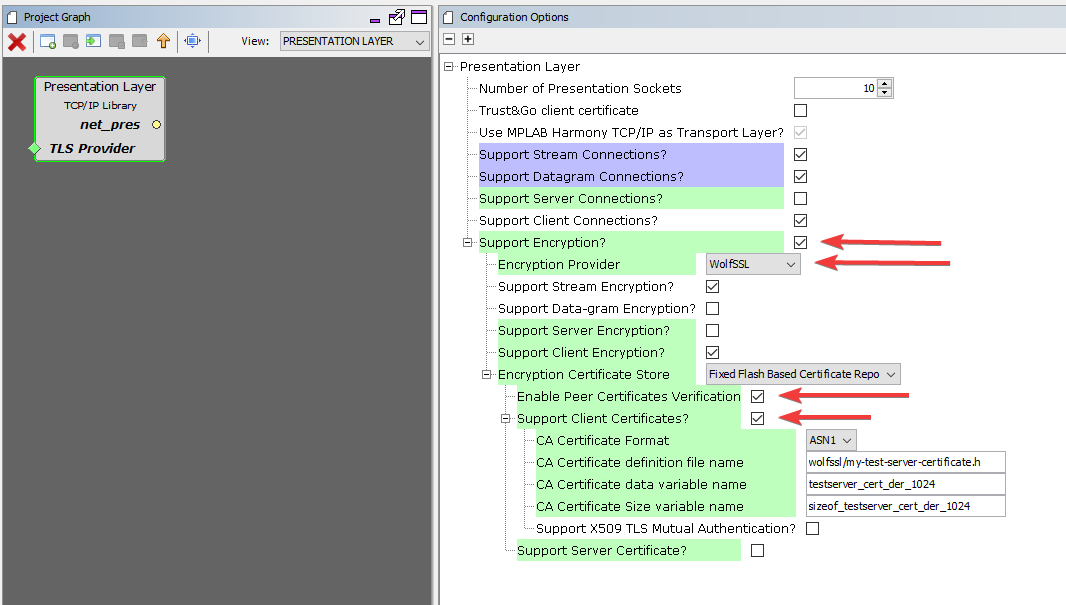


Figure Presentation Layer Configuration

1. In Harmony 3, we should add our header file in Presentation Layer. Kindly see how to add it below:

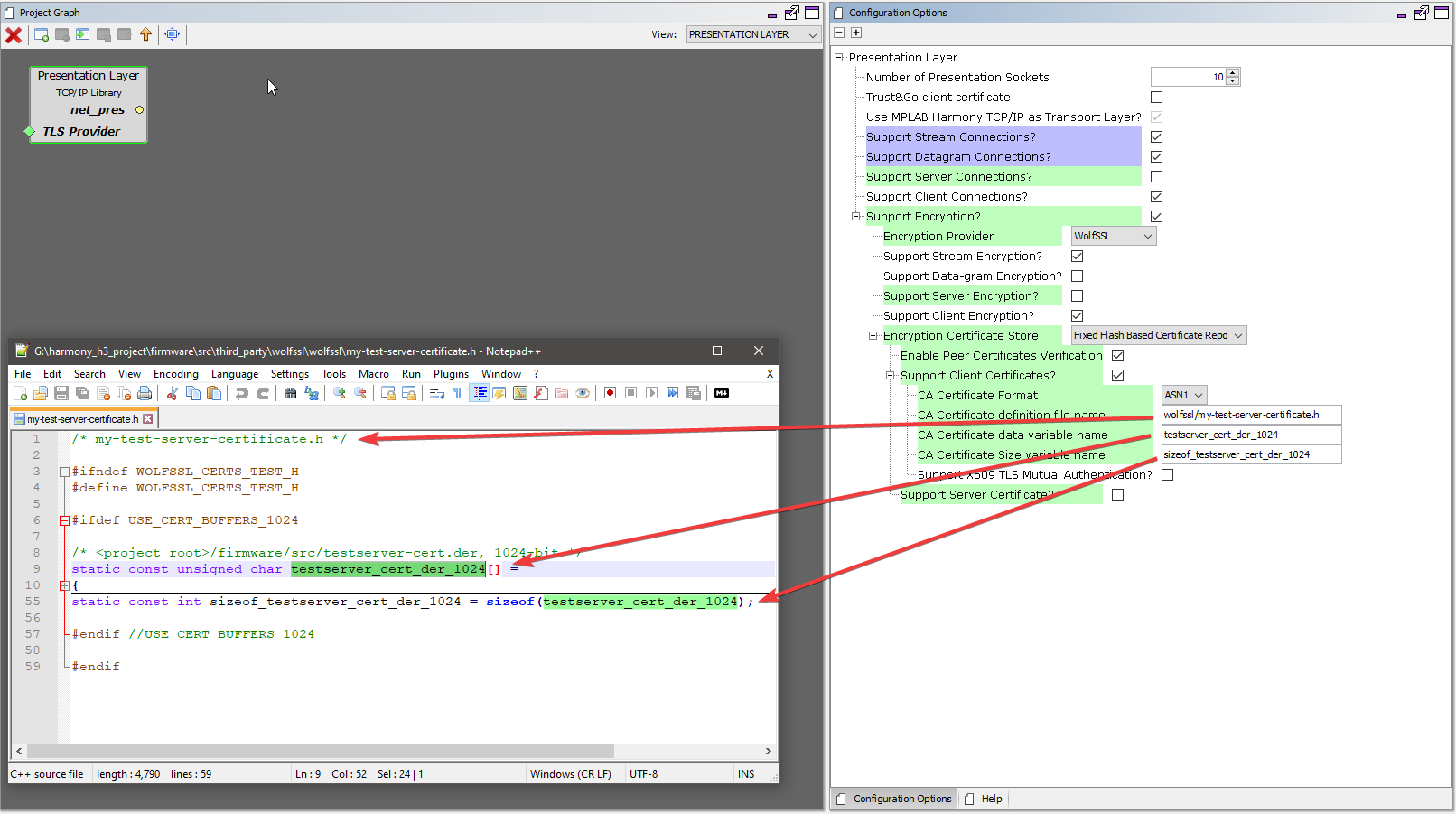


Figure Adding the Header File in Harmony V3

1. Define **USE\_CERT\_BUFFERS\_1024** in our own header file. Currently, Harmony TCPIP Presentation Layer uses a certificate buffer of 2048 by default.

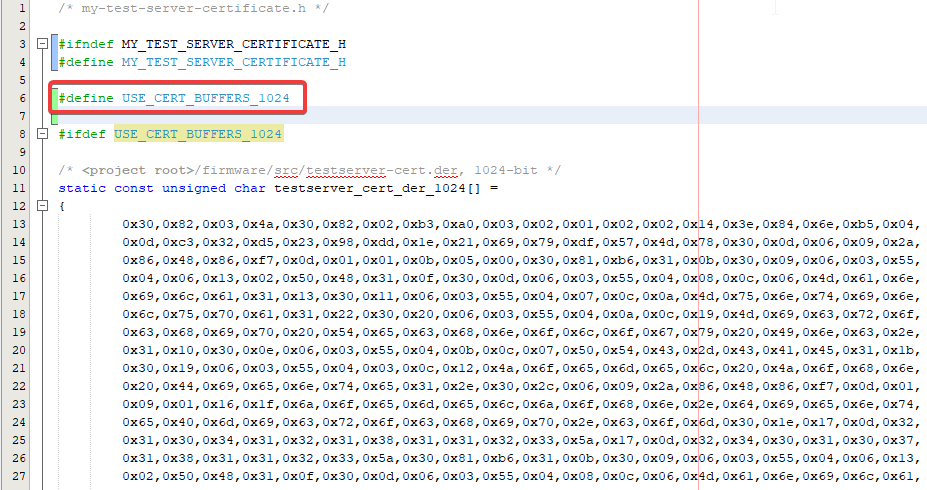


Figure WolfSSL Configuration for USE\_CERT\_BUFFERS\_1024

1. We have now finished adding our self-signed certificate to our client.
2. There are available demo code for WolfSSL TCP Client in Harmony V3 Net. For this training, we will be using the demo code, tcpip-secure.
3. The app.c, app.h, app\_commands.c and app\_commands.h should be copied in <Harmony V3 project dir>\firmware\src\.
4. Kindly add the files to MPLAB X IDE project tree.
5. That’s all set.We are now ready to run the demo code.

# Application Programming Interface

Below are the available APIs in Networking Presentation Layer of Harmony V3.

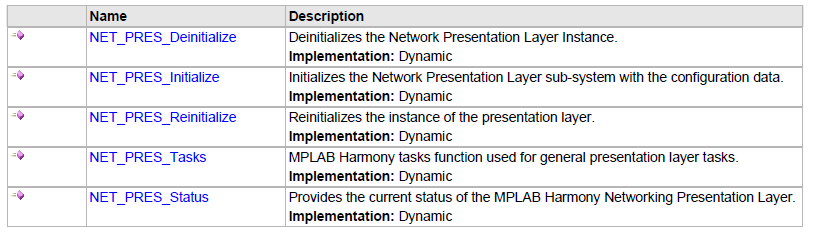
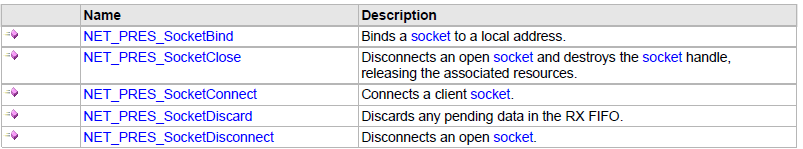


Figure System Functions



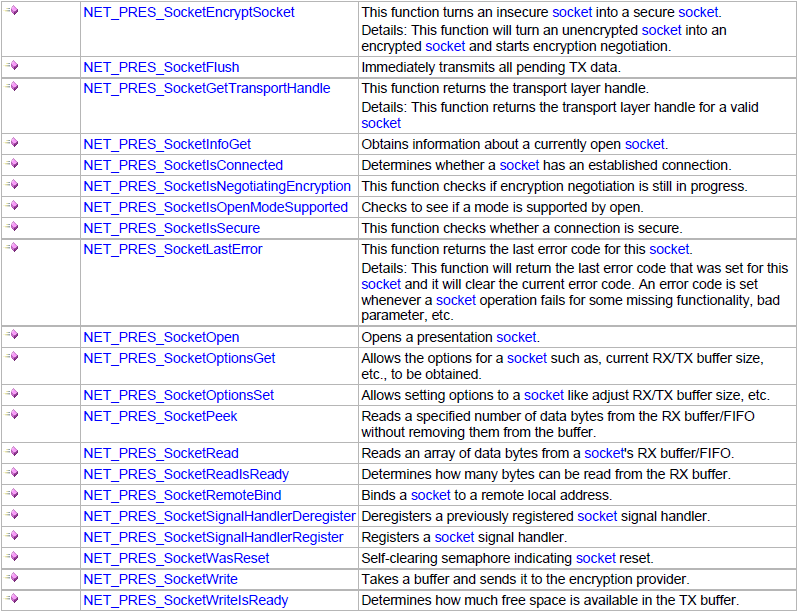


Figure Socket Functions

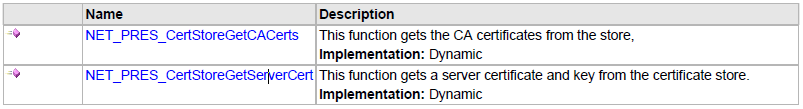


Figure Certificate Store Functions

The details of the functions above is discussed on the Harmony V3 Net Help Library.

# Example Code

This example code was forked from WolfSSL TCP Client demo from Harmony V3 Net Apps with changes to allow asynchronous communication. Asynchronous receiving of messages was made possible by Networking Presentation Socket Signal Handling API.

Currently, this example code only accept/processes for IPv4 server address only.

## Application Commands

User commands available on this example code are the following:

* “*connect\_tls <IP:Port>*” – This command opens a secure socket, connects to the server socket, then negotiates for secure communication. If negotiating the cipher and securing the socket was success, a signal handler is register to receive data asynchronously.
  + Parameter:
    - IP:Port – IPv4 address of the server + “:” + Server Port
  + Example:
    - “connect\_tls 192.168.137.2:65324”
      * IP – 192.168.137.2
      * Port – 65324
* “send\_msg <message>” – This command copies the argument variable to a buffer then writes it to a secure socket. This should be called after secure connection was established.
  + Parameter:
    - message – Client message to the server. The message should be enclosed in “ “. If not, only the first word will be sent.
  + Example:
    - “send\_msg “Hello, Server!”
* “disconnect\_tls” – This command closes the secure socket. This also de-registers the receive signal handler. This should be called after secure connection was established.
  + Parameter:
    - None.

## Sequence Diagram

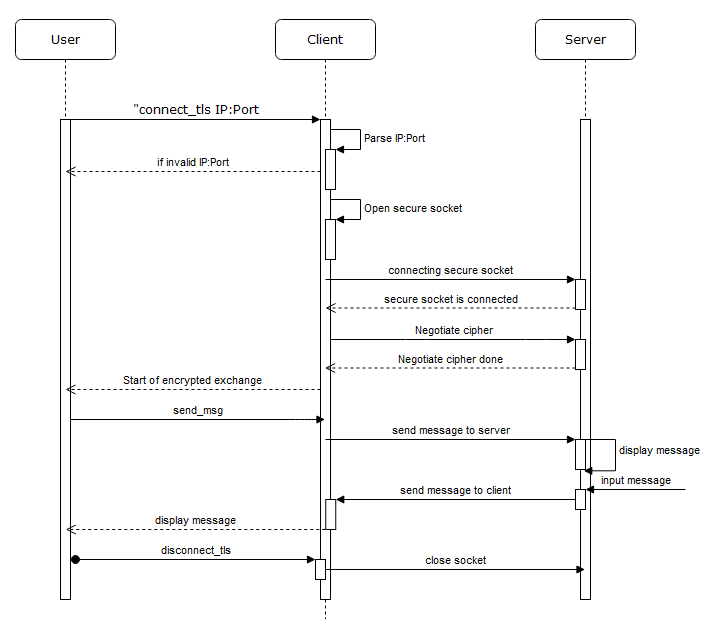


Figure Sequence Diagram (Client-side)

# Running the example code

To run the demo code, please open the serial port with baud rate of 115200.

Steps:

1. Type in “connect <ip:port>”. IP is the IPv4 address of the server and the port is listening port.
2. The console should prompt if the connection is secure and encrypted exchange of message is possible.
3. To send a message, use the command “send\_msg” followed by the message enclosed in quotation marks.
4. The demo code was designed to receive messages asynchronously so it will display immediately the message sent by the server.
5. To disconnect to the server, just simply type the command “disconnect\_tls”.

# Debugging

It is important to have access to the server and the client for ease of debugging. In Harmony V3 Framework, SSL/TLS related errors is displayed sent to the console by checking the settings below:

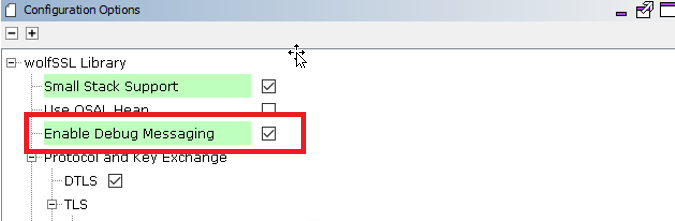


Figure 24 WolfSSL Enable Debug Messaging

The displayed error code coming from the console is a WolfSSL error code. Please see WolfSSL manual for more details or contact WolfSSL Team.

Another way to debug is to check SSL/TLS handshake using Wireshark. Below is an example of the complete SSL/TLS handshake in Wireshark.

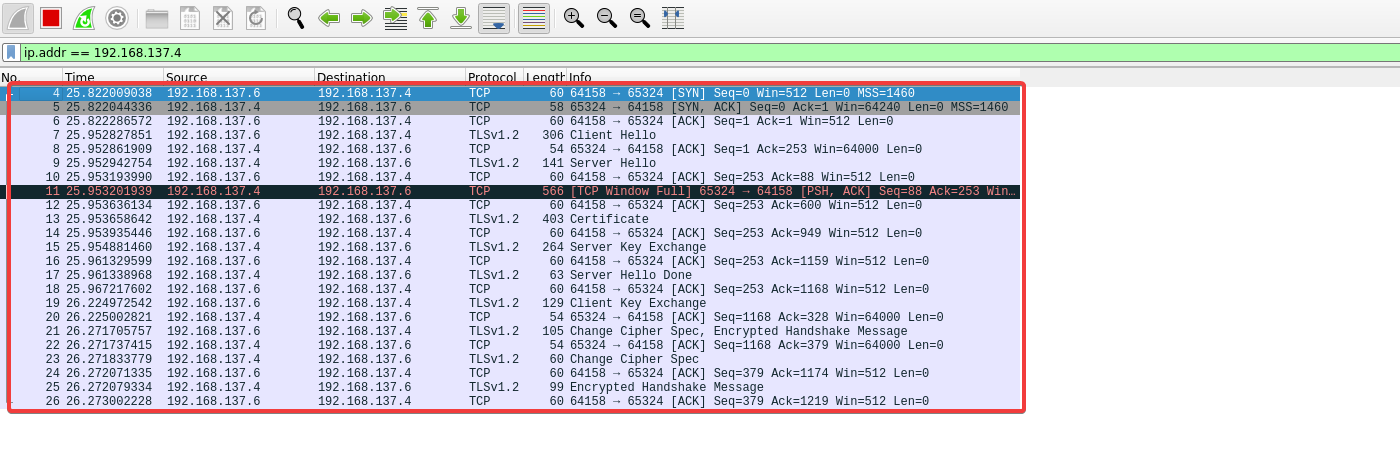


Figure 25 SSL/TLS Handshake in Action

This Wireshark log is very helpful in troubleshooting the handshake. It could tell where the handshake ended and also who ended the handshake.

# Learning Challenge

Create a secured TCP server using Harmony V3 for different architecture (like SAM devices, PIC32MX) with your own self-signed certificate.

# References

Creating Self-Signed Certificate

<https://github.com/wolfSSL/wolfssl-examples/blob/master/certs/taoCert.txt>

WolfSSL TLS Examples

<https://github.com/wolfSSL/wolfssl-examples/tls>

Bin2Hex

<https://www.browserling.com/tools/file-to-base64>

Dissecting TLS using WireShark

<https://blog.catchpoint.com/2017/05/12/dissecting-tls-using-wireshark/>