Evaluating PeerSec Networks' MatrixSSL on a Stellaris® Microcontroller

Application Note

AN01244-04

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com

Table of Contents

Introduction

Stellaris® microcontrollers offering Ethernet connectivity provide an excellent solution for intranet and internet control of embedded applications. In many circumstances, basic TCP/IP or HTTP protocols are adequate for control and status communication but in security-conscious applications, the additional protection offered by industry-standard Secure Sockets Layer (SSL) or its successor, Transport Layer Security (TLS) is desirable. This application note describes how one particular SSL software stack, MatrixSSL from PeerSec Networks, can be integrated with the StellarisWare® Peripheral Driver Library and the open source Lightweight IP (lwIP) stack to provide SSL support on a Stellaris® microcontroller.

The source code associated with this application note can be downloaded from the http://www.luminarymicro.com/products/software_updates.html web site. The package includes source for an SSL-enabled web server application and patches to the PeerSec Networks MatrixSSL code to allow operation on systems with no operating system running.

Software Licenses

MatrixSSL is available under two different software licenses. A GNU General Public License (GPL) version can be freely downloaded for evaluation purposes and for open-source project use. PeerSec Networks also licenses the code under a commercial license which does not carry the same restrictions as GPL and is, therefore, suitable for use in commercial or proprietary products where the source code is not published or where additional, non-GPL source is included.

As always, you are advised to study the license agreements of all software components incorporated into your application to ensure your compliance with those license agreements.

Acquiring, Installing, and Patching the MatrixSSL Source

The GPL version of MatrixSSL, which may be used for evaluation purposes and upon which the sample application described here is built, can be freely downloaded from http://www.matrixssl.org. At the time of writing, the current version of MatrixSSL is 1.8.6 and the download file is matrixssl-1-8-6-open.zip.

This version of MatrixSSL supports the following features:

- SSLv3 server and client
- RSA, 3DES, ARC4, SHA1, and MD5 crypto libraries
- RC4-MD5, RC4-SHA, and DES-CBC3-SHA cipher suites
- Full support for session resumption/caching
- Session rekeying and cipher renegotiation
- Server X.509 certificate chain authentication
- Parsing of .pem, DER, ASN.1, and X.509 formats
- PKCS#1.5 and PKCS#5 support for key formatting
- Small code size (approximately 50 KB including crypto providers)

The commercial version of MatrixSSL adds the following features:

- TLS
- AES crypto library
- AES128-SHA cipher suite
- Client X.509 certificate chain authentication
- SSH command line support

The patch file provided with the sample web server application assumes you are using the GPL version of MatrixSSL 1.8.6. If you are using the commercial version, it is expected that the same patches will apply but you are encouraged to apply them to the GPL version first and then merge the changes manually with the commercial source since this has not been tested.

If you have not done so already, ensure that you have the Stellaris Firmware Development Package installed on your system prior to continuing. This can be downloaded from http://www.luminarymicro.com/products/software_updates.html as a self-extracting ZIP file. Note that this sample requires release 3618 or later. If you last updated your source prior to October 2008, you will need to download and install the latest version to ensure that the application described here can build and run correctly.

After downloading the matrixssl-1-8-6-open.zip file, unzip it into a directory named MatrixSSL at the same level as your StellarisWare® tree. If you installed StellarisWare® in the default directory, C:\StellarisWare, then this implies that you should unzip the MatrixSSL package into C:\MatrixSSL. If this is done correctly, you will find a directory named C:\MatrixSSL\matrixssl-1-8-6-open on your system. Note that the build process assumes this location for MatrixSSL relative to StellarisWare™, so if you move the MatrixSSL code to a different subdirectory level, you will need to modify the application note source and makefiles to reflect this.

Download the application note source code from the software update web site: http://www.luminarymicro.com/products/software_updates.html and unzip the file into the directory above StellarisWare®. If you used the default directory, this will be C:\. You should find that this adds directory StellarisWare\AppNotes\sw01244 (among several others) if performed correctly. You should also find a patch file, matrixssl-1-8-6-open.patch, written to the directory above StellarisWare, typically C:\.

Apply the patch to the MatrixSSL source tree as follows. This assumes default installation directories, so adjust these as necessary if you installed elsewhere. If you typically perform builds using an IDE interface rather than the command line, note that the patch step is not performed by the project and workspace files included in the application note source. You will, therefore, have to perform this step manually from a command line. If using Code Red Technologies' Red Suite tools, you must perform this step before importing the application source into your workspace (as described later).

1. Start the Cygwin or UnxUtils shell you typically use to build applications for Stellaris platforms. Ensure that you have the "patch" application available in your path. If you do not have access to patch, you can install it via Cygwin setup or download it from http://gnuwin32.sourceforge.net/packages/patch.htm.

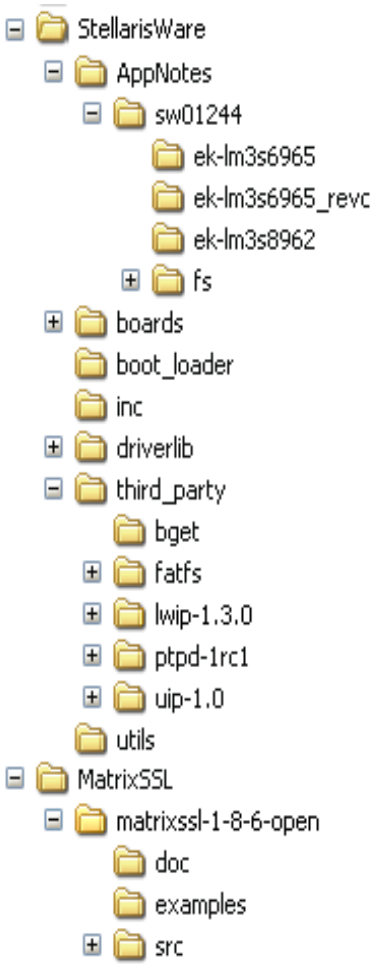
2. Change directory to C:\StellarisWare\AppNotes\sw01244 (or /cygdrive/c/StellarisWare/AppNotes/sw01244 if using Cygwin).
3. Execute "make prep".

On successful application of the patch, the following should display on the screen:

```
patch -Nup1 -d ../../../MatrixSSL/matrixssl-1-8-6-open -i ../../matrixssl-1-8-6-open.patch
patching file `src/Makefile'
patching file `src/matrixConfig.h'
patching file `src/matrixSsl.c'
patching file `src/os/debug.c'
patching file `src/os/no_os/no_os.c'
patching file `src/os/osLayer.h'
patching file `src/os/psMalloc.h'
```

You have now installed and patched all the source code necessary to move on and build the sample web server application. At this stage, your source tree should contain the following:

Figure 1. DriverLib and MatrixSSL Source Trees



Building the Example Web Server Application

The sample web server application includes command line makefiles for use with all supported tool chains, and workspace/project files for IAR Embedded Workbench, Keil µVision3, and Code Red Technologies' Red Suite IDEs. Support is provided for the following Stellaris evaluation kits:

- LM3S8962 Ethernet + CAN Evaluation Kit
- LM3S6965 Ethernet Evaluation Kit

Two versions of the application are provided for the LM3S6965 Evaluation Kit to support the different displays found on revisions A and C of the board. You can determine which of these boards you have from the silk-screen printing on the reverse side of the PCB, beneath the JTAG connector.

Building from the Command Line

1. Start your usual Cygwin or UnxUtils command shell and change to the DriverLib source directory, typically, C:\StellarisWare\driverlib (or /cygdrive/c/StellarisWare/DriverLib).
2. Execute the `make` command to build the DriverLib library with your default toolchain (typically arm-stellaris-eabi-gcc). If you are using a different toolchain, you can either set the COMPILER environment variable or pass the variable as a parameter to make. Valid values for COMPILER are `gcc` (when using the free distribution of GNU tools for Stellaris), `sourcerygxx`, (when using CodeSourcery's distribution of gcc including their specific startup and library code), or codered (when using the gcc version shipped with Code Red Technologies' Red Suite toolchain).
3. Change to the directory containing the application note source. Typically, this is C:\StellarisWare\AppNotes\sw01244 (or /cygdrive/c/StellarisWare/AppNotes/sw01244).
4. Execute the `make` command once more to build the sample application and link with the DriverLib library built in Step 2. You should expect no errors during the build but beware that warnings in third-party code have not been fixed so the build output will not be completely clean.
5. Once the build completes, executables for each target board can be found in the ek-lm3sxxxx/${COMPILER} directory, for example, ek-lm3s8962/gcc. The ELF format executable is named webserver-ssl-lm3sxxxx.axf where xxxx contains the target part number. An equivalent binary image named webserver-ssl-lm3sxxxx.bin is also created in the same directory.

Executing the `make` command from the sw01244 directory will create executables for all supported evaluation kit boards. To build for a single board, change into the relevant ek-lm3sxxxx subdirectory and execute the `make` command from there.

The binary image created by the build process can be downloaded to the Stellaris flash using the LM Flash Programmer application (available for download from http://www.luminarymicro.com/products/software_updates.html). Alternatively, you can also flash the image from the AXF-format output file using a debugger such as gdb.

Building Using IAR Embedded Workbench

Workspace files which will build DriverLib and the three versions of the web server application can be found in the application note source directory as webserver-ssl.eww (for IAR Embedded Workbench, version 5.1) and webserver-ssl-ewarm4.eww (for IAR Embedded Workbench, version 4.2). Individual project files can also be found in each of the board-specific subdirectories allowing each board's

version to be built independently. Workspace and project files that contain "-ewarm4" are for IAR Embedded Workbench 4.2. The other workspace and project files are for IAR Embedded Workbench 5.1.

To build all the targets, do the following from within the IAR Embedded Workbench IDE:

1. Select File/Open/Workspace… from the IDE menu.
2. Navigate to the application note directory, typically, C:\StellarisWare\AppNotes\sw01244, and select the webserver-ssl.eww file, then click the Open button.
3. Right -click one of the three target applications you can see in the Workspace window and then select "Set Active" from the context menu.
4. Select Project/Make from the IDE menu or press F7 to start building the active application.
5. Repeat Steps 3 and 4 for other versions of the application that you wish to build.

The image created by the build may be flashed to the target evaluation kit board by selecting Project/Debug from the IDE menu.

Building Using Keil µVision3

A workspace file which will build DriverLib and the three versions of the web server application can be found in the application note source directory as webserver-ssl.mpw. Individual project files for Keil's µVision3 IDE are provided in each of the ek-lm3sxxxx directories in the DriverLib\AppNotes\sw01244 directory, and these may be used to build the sample web server for your specific evaluation kit board.

To build a single target executable, do the following from within the Keil µVision3 IDE:

1. Select Project/Open Project… from the IDE menu.
2. Navigate to the application note directory, typically, C:\StellarisWare\AppNotes\sw01244, then change into the directory whose name describes the evaluation kit board you are building the application to target, ek-lm3s8962, ek-lm3s6965, or ek-lm3s6965_revc.
3. Click the single uv2 file you find in the directory and click the Open button.
4. Select Project/Rebuild all target files from the IDE menu.
5. Repeat steps 2-4 for other target boards you may be using.

Once the executable has been built, it may be downloaded to the board using the Flash/Download option on the IDE menu. Alternatively, the binary image found in the StellarisWare\AppNotes\sw01244\ek-lm3sxxxx\rvmdk directory may be written to the target board using the LM Flash Programmer application (available for download from http://www.luminarymicro.com/products/software_updates.html).

Building Using Code Red Technologies' Red Suite

An XML project file is provided to allow you to import the webserver-ssl project into Code Red Technologies' Red Suite IDE. Individual project description files are provided in each of the ek-lm3sxxxx directories in the DriverLib\AppNotes\sw01244 directory, and these may be used to import the required source code into the Red Suite workspace and to build the application for your specific evaluation kit board.

The following instructions assume that you have already imported the StellarisWare™ Firmware Development Package for your target board into the Red Suite IDE (for information on how to do this, see the Quickstart document from the evaluation kit CD), and that you have installed and patched the MatrixSSL evaluation release as detailed earlier in this document.

To import and build the webserver-ssl application, do the following from within the Red Suite IDE:

1. From the Quickstart Panel, expand the "Projects and Files" list, then click "Import project(s) from XML description".
2. Click the "Browse" button on the "Create project from resources" dialog box that appears.
3. Navigate to the application note example directory for your target board and select the cr_project.xml file you find there. You must select the file for the correct board—if you chose a file for a board other than the one your workspace targets, the application will either not build or will show unexpected results when you attempt to run it. If you unzipped the application note software using the default directory, the cr_project.xml file will be found in C:\StellarisWare\AppNotes\sw01244\ek-lm3sxxxx where xxxx represents the particular target board.
4. Click "Finish" on the "Create project from resources" dialog box. Red Suite will import the example and build it. Once this is complete, you should see output in the console window summarizing the sizes of the various application segments.

Once the application has been built, it may be downloaded to the board using the "Flash Utilities" which are accessible from the Red Suite toolbar or via the Red Suite debugger. To download and run the application via the debugger, ensure that your evaluation board is connected to your PC via the USB cable, then click the "webserver-ssl-ek-lm3sxxxx" project in the Project Explorer window, expand the "Debug and Run" list in the Quickstart Panel, and click "Debug project 'webserver-ssl-ek-lm3sxxxx'" to flash the application to your board and start the debugger.

Running the Web Server Application

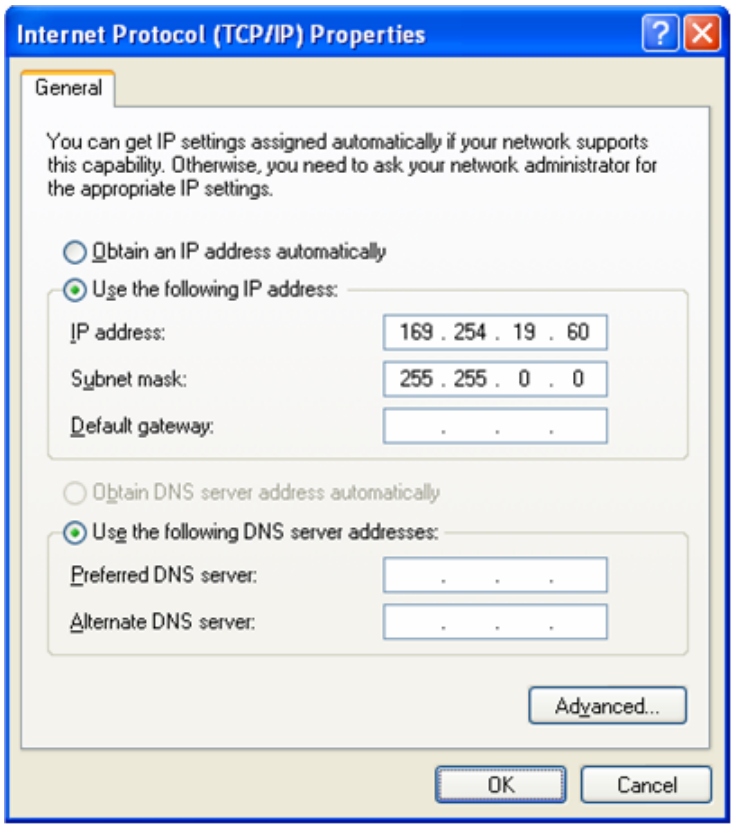
The webserver-ssl sample application allows you to connect to an EK-LM3S8962 or EK-LM3S6965 board using secure URLs from a standard web browser. The application handles SSL session management and serves files from either a small internal file system or an installed microSD card via the secure connection.

When initially started, the application attempts to acquire an IP address from a DHCP server. If this process times out, a link local address (169.254.*.*) is chosen using the algorithm specified in internet standard RFC3927 which is also known as AUTOIP, IPv4 Link-Local (IPV4LL), Automatic Private IP Addressing (APIPA), or Internet Protocol Automatic Configuration (IPAC). This system allows the board to be tested either on a normal Ethernet or in a back-to-back configuration where the board is attached directly to a PC with no DHCP server present.

If the DHCP attempt fails and a link local IP address is used, then the PC must be configured to be on the same subnet as the board. In most cases, the PC detects the correct IP address and subnet settings automatically after several seconds. In some cases, the PC's IP address and subnet mask must be configured manually. To do this, disable the PC's wireless network connection and any other internet connections that could interfere with the network being created. Select the Internet Protocol (TCP/IP) connection within the Local Area Connection Properties and click on Properties. Next,

manually configure the PC's IP address to a different link local address from that chosen by the Stellaris board (for example 169.254.19.60) and subnet mask to 255.255.0.0, as shown in Figure 2.

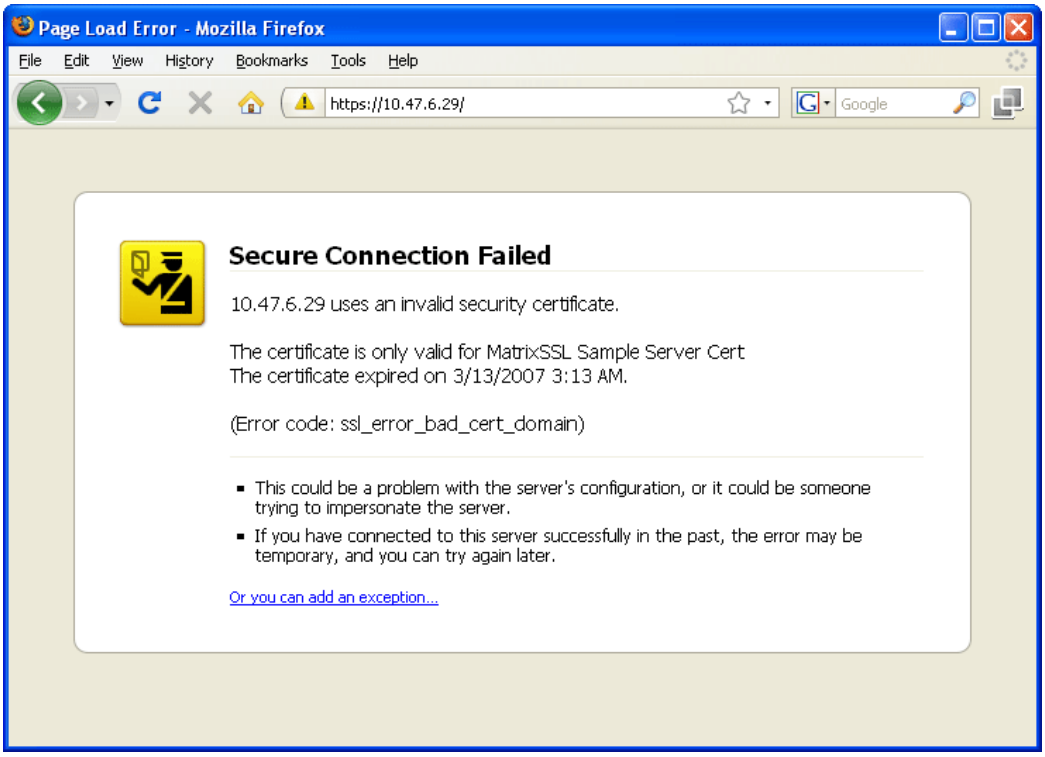
Figure 2. TCP/IP Properties



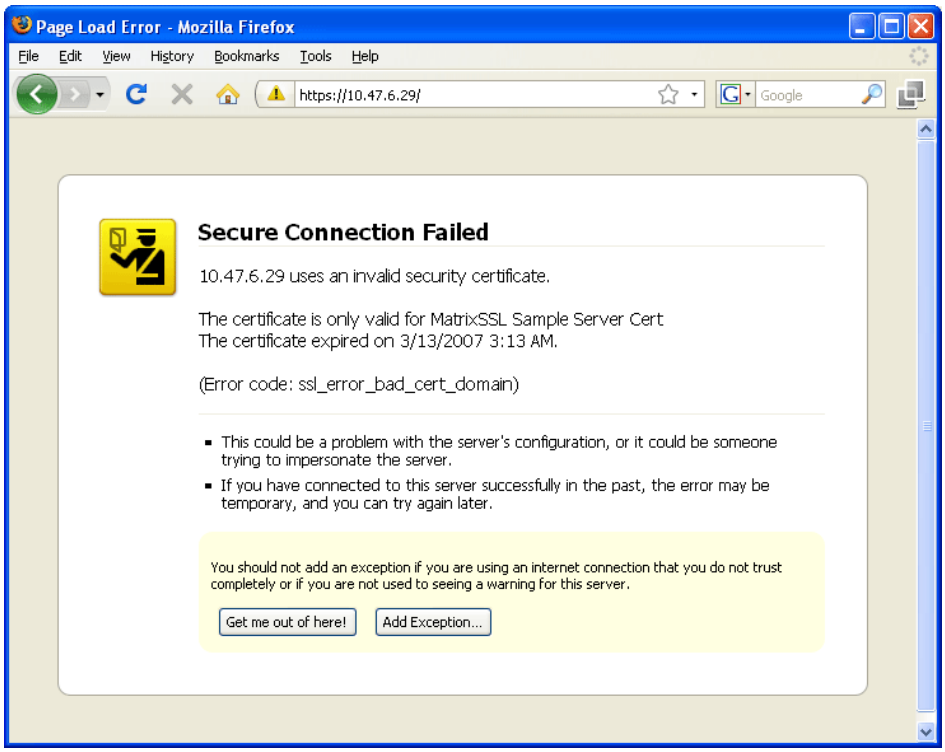
Once the IP address has been configured, it is shown on the OLED display. At this point, you can run the web browser of your choice and access the web site served by the board by entering URL "https://<board_ip_address>", for example, https://10.47.6.14 in the example below. The example does not support http access.

The first time you attempt to access the sample web server at a particular IP address, you will receive a warning from the browser about the validity of the certificate presented. This is due to the fact that the web server application uses a self-signed test certificate and it is safe to continue past this warning. Depending on the browser you are using, you may have to indicate that you wish to accept the certificate regardless of the fact that it has expired and is for a different site. The following example shows the steps that you should take to install a security exception using Firefox 3.0.

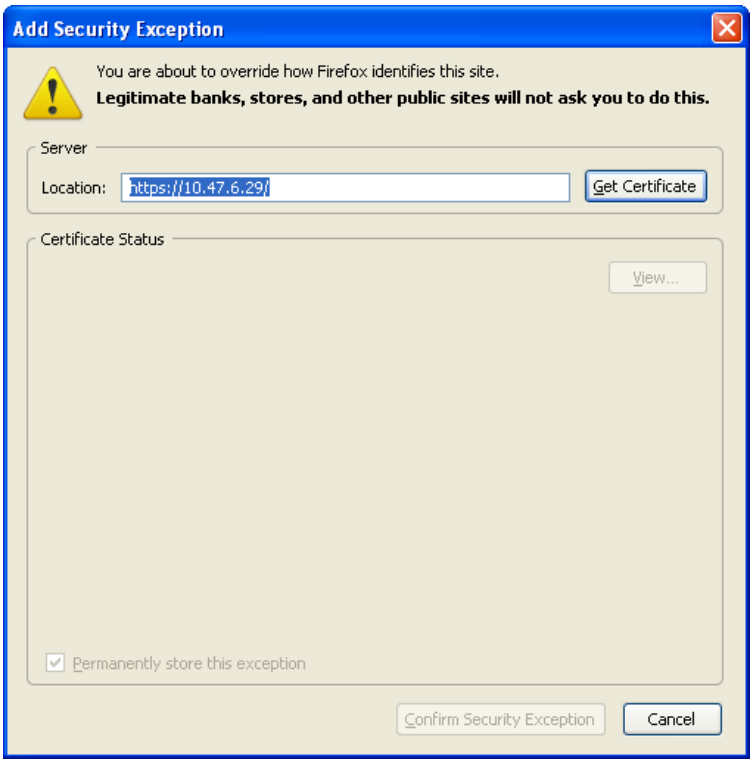
On your initial attempt to connect to the evaluation kit board, you will see the following:



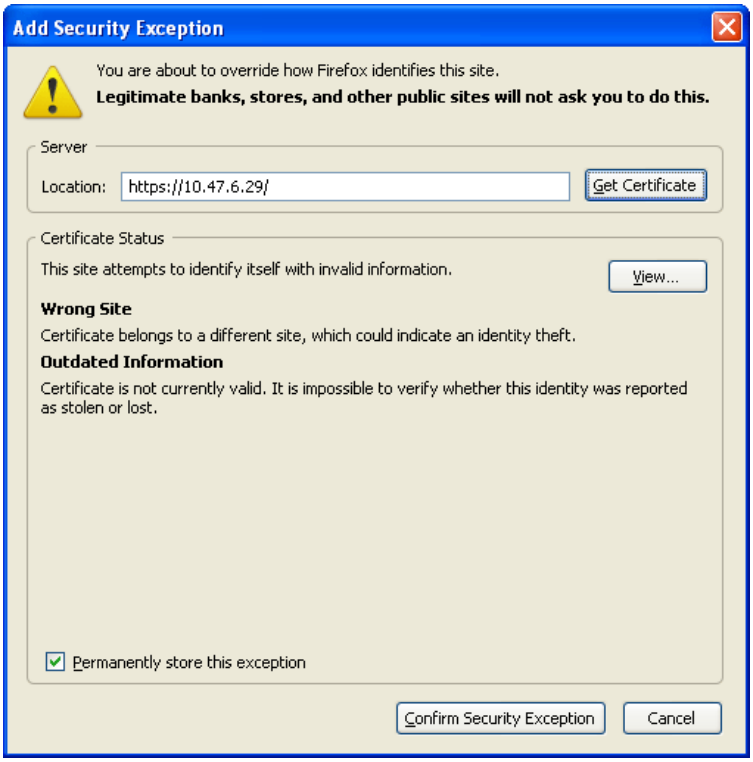
Click "Or you can add an exception..." which will change the display to:



Click "Add Exception..." to show the "Add Security Exception" dialog box



then press the "Get Certificate" button which should result in the following:



Click "Confirm Security Exception" and the secure connection should be allowed.

On successful connection, the following page displays:

Figure 3. Stellaris Secure Web Server Page

FireFox and Internet Explorer each set up multiple (typically three), simultaneous SSL sessions with the server to serve the pages from the internal file system site. Note that a single SSL connection can require up to 16 KB of buffering depending on the data to be transferred and it is easily possible to run out of RAM on the evaluation board by constructing pages which cause the browser to attempt to open more sessions than can be accommodated. If this occurs, you will see objects missing from the page.

Application Internals

While a general-purpose SSL web server may be a tall order for a system with a total of 64 KB of SRAM, the web server application does offer an easily verifiable SSL implementation which does not require any custom PC application to act as the other end of the SSL connection. The code is also structured in such a way that the SSL layer is isolated from the HTTP server, therefore allowing easy reuse of the SSL code within other applications running their own protocols on top of it.

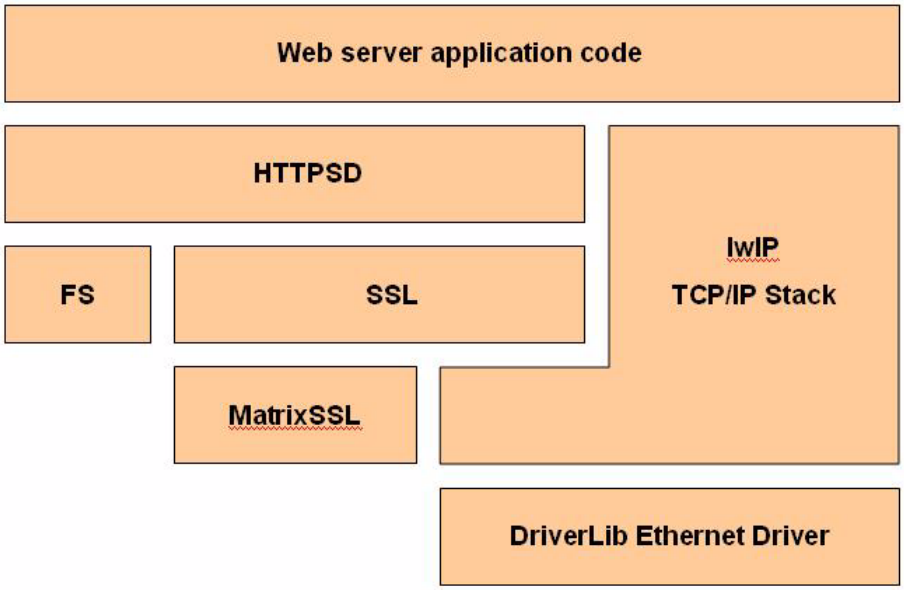
The sample application consists of six main blocks:

- Web server application code
- lwIP: An open source light-weight IP stack
- MatrixSSL: PeerSec Network's compact SSL library

- SSL layer: An SSL API mirroring lwIP's TCP API allowing easy porting of the existing lwIP HTTP server above the new SSL layer
- HTTPSD: Secure HTTP server layer
- FAT file system support module

No operating system is used in this implementation.

Figure 4. Application Block Diagram



Web Server Application Code

The webserver-ssl.c file contains the main application entry function, initialization code, and the loop which controls the IP stack and, ultimately, all operations carried out by the application.

Application timing is controlled by the SysTick interrupt which is set to fire every 10 mS resulting in a call to lwip_tick which makes the appropriate management function calls into lwIP at the required intervals.

This file also contains the low-level Ethernet interrupt handler which simply clears the source of the interrupt, disables the interrupt, and sets flags which will cause lwip_tick to be called to process any incoming packets and reenable the interrupt.

lwIP (StellarisWare/third_party/lwip_1.3.0)

Light-weight IP (lwIP) is an open source TCP/IP stack designed for use on microcontrollers or other memory-constrained platforms.

The lwIP release includes an HTTPD server which is used in the enet_lwip sample application shipped with the DriverLib release. This code is used as the basis for the HTTPSD block within webserver-ssl.

MatrixSSL

This block represents all software under MatrixSSL/matrixssl_1_8_6_open forming the low-level SSL support blocks of the application. The Matrix library handles all session management handshaking and encryption/decryption of data for each HTTPS request.

The version of the code used here contains patches to allow operation in an environment with no operating system and to enable debug output (when enabled) via Stellaris UART0.

SSL (ssl.c)

To allow easy retargeting of the existing lwIP HTTPD module as an HTTPSD server, the SSL layer was developed to mirror the lwIP TCP API. Each function offered at the TCP layer is implemented in ssl.c in such a way as to hide the SSL specifics from the layer above. Using this remapping, an existing layer using the lwIP TCP API can be retargeted to use a secure connection merely by replacing the tcp_ prefix on all its lwIP function calls with ssl_.

Many of the functions offered by the SSL layer pass directly to the equivalent function in the lwIP TCP layer. Those that differ significantly relate to acceptance of incoming connection requests (via ssl_tcp_accept), reception of data (via ssl_tcp_recv), and transmission of data (via ssl_write and ssl_tcp_sent) which call the MatrixSSL library to perform encode and decode operations.

The MatrixSSL API cleverly encapsulates the client/server handshake sequence performed during connection initiation in such a way that the code using the library can call sslDecode for all data received and then perform any required follow-up actions based on the return code from the function. Such actions may include sending a block of data back to the client or passing decoded data to a higher layer. This operation is handled within ssl.c during the ssl_tcp_recv function which is called by lwIP for every TCP segment received.

HTTPSD (httpsd.c)

As mentioned previously, this file contains a version of the lwIP httpd.c HTTPD server module which has been retargeted for use the SSL layer rather than the lwIP TCP API. Changes here include renaming of all the previous tcp_ APIs to call their SSL counterparts.

FAT File System (StellarisWare/third_party/fs)

The sample application makes use of either a small internal FAT file system image or a FAT file system on an inserted microSD card. Code to support this file system can be found under StellarisWare/third_party/fs. This code is also used by various other sample applications provided alongside DriverLib.

Modifications Made to the MatrixSSL Source Tree

The MatrixSSL patch supplied with this application note is mainly concerned with adding support for systems without any operating system and hooking the MatrixSSL debug features to the DriverLib uartstdio library. No core operational changes are made to the MatrixSSL code. The patch contains the following changes:

- "Makefile" on page 16
- "matrixConfig.h" on page 16

Makefile

As shipped, the Makefile for MatrixSSL will generate various libraries but does not support the various toolchains that DriverLib supports. The patched Makefile, essentially a replacement, allows use of the DriverLib command-line `make` process to generate a MatrixSSL library.

Note that the sample application does not make use of this Makefile but builds the MatrixSSL source files directly alongside the other application code. This file is, therefore, included as a convenience to anyone who wishes to build the MatrixSSL code independently of the application.

matrixConfig.h

This header file contains various configuration parameters use by MatrixSSL code. The patches applied to it define the amount of RAM to set aside for use by SSL (`MAX_SSL_HEAP_SIZE`) and also include several specific header files related to debug output.

Additional changes are intended to trim the memory usage and image size. When built for non-Windows platforms, the following changes are made:

- The SSL session cache is reduced from 32 to 8 entries
- Support for RSA with triple DES is removed
- Multi-thread support is disabled
- File system support is disabled
- 64-bit integer support is disabled

The last of these changes, which does decrease performance somewhat, was implemented due to a problem observed when using IAR Workbench at its highest optimization level. In this case, with "long long" support enabled, the multiple-precision integer support functions in the mpi.c file do not operate correctly which results in SSL handshake failure. This problem has not been observed with other toolchains.

matrixSsl.c

Since we replace the C runtime library time() function with a local version, this file is patched to remove system header <time.h> when building for a Stellaris target.

debug.c

References to C runtime function `printf()` have been patched to call the DriverLib function `UARTprintf` instead. This ensures that debug output is sent via UART0 on the evaluation kit board and prevents inclusion of an unnecessary part of the C runtime library.

no_os.c

This new file provides an implementation of the MatrixSSL OS API (defined in osLayer.h) that allows the software to run without any operating system. Functions in this file handle memory heap management on behalf of MatrixSSL (using the open source bget heap manager), provide a replacement for the C runtime library time() function, and generate blocks of pseudo-random data.

Note that this file is derived from GPL source provided within the MatrixSSL package but includes bget and DriverLib headers—components not covered by GPL. As a result, this implementation may only be used for evaluation purposes and cannot be shipped in a commercial product unless a non-GPL version of the MatrixSSL software is used.

osLayer.h

This file defines the OS-like functions used by the MatrixSSL software and implemented in the no_os.c file. Changes from the basic version shipped with MatrixSSL are as follow:

- Redefine `time()` to `NoOSTime()` to prevent pulling in the C runtime function.
- Redefine `sslAssert` to target output to the `UARTprintf` function rather than C runtime `fprintf`.

psMalloc.h

This header defines the memory manager interface used by MatrixSSL. The original version maps the various functions to the C runtime equivalents, for example `psMalloc()` is mapped to `malloc()` and `psFree()` to `free()`. To prevent the need to pull in the C runtime support, these are redefined to functions exported by no_os.c which use the compact bget heap manager instead.

Debug Features

By default, the web server application will build with debug features disabled to provide minimum code size and maximum performance. Both MatrixSSL and lwIP do, however, contain debug features that greatly aid development and problem diagnosis.

MatrixSSL

Debug output from the MatrixSSL software can be enabled by defining both "DEBUG" and "MATRIXSSL_DEBUG=1" labels passed to the compiler preprocessor. This can be accomplished by editing Makefile in the sw01244/ek-lm3sxxxx directory and appending:

```
-DDEBUG -DMATRIXSSL_DEBUG=1
```

to labels CFLAGSgcc and CFLAGSsourcerygxx and then rebuilding.

If using either the IAR Workbench or Keil µVision3 IDE, these labels can be defined from within the relevant project options or properties dialog box.

lwIP

To enable debug features in lwIP, edit the lwipopts.h file found in the sw01244 directory. Under the "Debugging Options" section, you will find labels for each of the lwIP functional areas. Defining a

label as LWIP_DBG_ON will enable trace output for that function, assuming LWIP_DEBUG and DEBUG are also defined.

A debug build of lwIP will also contain a global structure named lwip_stats which contains useful system throughput information including, memory usage, and errors detected. This structure is particularly helpful when tuning for memory usage since it keeps track of the maximum allocation from each of the lwIP buffer pools.

DriverLib

Debug features of DriverLib and the webserver-ssl application itself may be enabled by ensuring that DEBUG is defined during the build. As above, this can be most easily accomplished by editing Makefile in the appropriate evaluation kit board directory below sw01244.

Conclusion

The example secure web server described in this application note shows that Stellaris microcontrollers offer a suitable platform for the development of SSL- or TLS-based, networked applications. The ARM Cortex-M3 CPU running at 50 MHz provides sufficient bandwidth to implement the complex cryptography algorithms required without compromising user responsiveness and the binary image size of approximately 100 KB leaves up to 156 KB of flash remaining (depending on the specific Stellaris microcontroller in use) for other application-specific modules.

References

The following documents and source code are available for download at www.luminarymicro.com:

- *StellarisWare™ Peripheral Driver Library User's Guide*, Publication Number SW-DRL-UG
- StellarisWare™ Software, Order Number SW-LM3S
- LMFlashProgrammer, GUI-based flash programmer application

The following web resources related to the software and protocols used by this example may be of interest:

Protocols

SSLv2	http://wp.netscape.com/eng/security/SSL_2.html
SSLv3	http://wp.netscape.com/eng/ssl3/draft302.txt
TLS 1.0	http://rfc.dotsrc.org/rfc/rfc2246.html http://en.wikipedia.org/wiki/Transport_Layer_Security
RFC3927 (AUTOIP)	http://tools.ietf.org/html/rfc3927

Software Modules

lwIP	http://lwip.scribblewiki.com/LwIP_Main_Page
MatrixSSL (GPL license)	http://www.matrixssl.org

MatrixSSL (Commercial license)
http://www.peersec.com/matrixssl.html

General Information

The TCP/IP Guide http://www.tcpipguide.com/free/index.htm

Important Notice

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265