CyaSSL **Porting Guide**

Version 1.4
November 7, 2013

Purpose

This guide provides a reference for developers and engineers porting the CyaSSL embedded SSL library to new embedded platforms, operating systems, or transport mediums (TCP/IP, bluetooth, etc.). It calls out areas in the CyaSSL codebase which typically require modification when porting CyaSSL. It should be considered a "guide" and as such, it is an evolving work. If there is something you find missing, please let us know and we'll be happy to add instructions or clarification to the document. One of our main goals for CyaSSL is ease of use.

Audience

This guide caters to developers or engineers porting the CyaSSL embedded SSL library to new platforms or environments that are not supported by default.

Table of Contents

This table of contents includes commentary on when each section needs to be read. Hopefully, this will expedite the reading process and eliminate unnecessary work.

1. Introduction

Several steps need to be iterated through when getting CyaSSL to run on an embedded platform. Some of these steps are outlined in Section 2.4 of the CyaSSL Manual ("Building in a non-standard environment": http://yassl.com/yaSSL/Docs-cyassl-manual-2-building-cyassl.html).

Apart from steps in Chapter 2 of the CyaSSL Manual, there are several areas in the code which may need porting or modifications in order to accommodate a specific platform. CyaSSL abstracts many of these areas - attempting to make it as easy as possible to port CyaSSL to a new platform.

In the ./cyassl/ctaocrypt/settings.h file, there are several defines specific to different operating systems, TCP/IP stacks, and chipsets (MBED, FREESCALE_MQX, MICROCHIP_PIC32, MICRIUM, EBSNET, etc.). New defines are typically added to the settings.h file when a new port of CyaSSL is completed which requires modifications to CyaSSL. It provides an easy way to turn on/off features as well as customize build settings. Feel free to add a new custom define in this file when doing a port of CyaSSL to a new platform. We encourage users to contribute ports of CyaSSL back to the main open source code branch. This helps keep CyaSSL up to date and allows different ports to remain updated as the CyaSSL project improves and moves forward.

wolfSSL encourages the submission of patches and code changes through either direct email (info@wolfssl.com), or through GitHub pull request (https://github.com/cyassl/cyassl).

2. Porting CyaSSL

2.1 Data Types

Q: When do I need to read this section?
A: Setting the correct data type size for your platform is always important.

CyaSSL benefits speed-wise from having a 64-bit type available. Set **SIZEOF_LONG** or **SIZEOF_LONG_LONG** to match the result of sizeof(long) and sizeof(long long) on your platform. This can be added to your custom define in the settings.h file. For example, in settings.h under a sample define of MY_NEW_PLATFORM:

```
#ifdef MY_NEW_PLATFORM
      #define SIZEOF_LONG 4
      #define SIZEOF_LONG_LONG 8
      ...
#endif
```

2.2 Endianness

Q: When do I need to read this section?
A: Your platform is a big endian system.

Is your platform big endian or little endian? CyaSSL defaults to a little endian system. If your system is big endian, define **BIG_ENDIAN_ORDER** when building CyaSSL. Example of setting this in settings.h:

```
#ifdef MY_NEW_PLATFORM
      ...
      #define BIG_ENDIAN_ORDER
      ...
#endif
```

2.3 writev

Q: When do I need to read this section?
A: <sys/uio.h> is not available.

By default, the CyaSSL API makes available CyaSSL_writev() to applications, which simulates writev() semantics. On systems that don't have the *<sys/uio.h>* header available, define **NO_WRITEV** to exclude this feature.

2.4 Input / Output

Q: When do I need to read this section?
A: A BSD-style socket API is not available, you are using a custom transport layer or TCP/IP stack, or only want to use static buffers.

CyaSSL defaults to using a BSD-style socket interface. If your transport layer provides a BSD socket interface, CyaSSL should integrate into it as-is, unless custom headers are needed.

CyaSSL provides a custom I/O abstraction layer which allows users to tailor CyaSSL's I/O functionality to their system. Full details can be found in Section 5.1.2 of the CyaSSL Manual:
http://yassl.com/yaSSL/Docs-cyassl-manual-5-portability.html

Simply put, you can define CYASSL_USER_IO, then write your own I/O callback functions using CyaSSL's default EmbedSend() and EmbedReceive() as templates. These two functions are located in ./src/io.c.

CyaSSL uses dynamic buffers for input and output, which default to 0 bytes. If an input record is received that is greater in size than the buffer, then a dynamic buffer is temporarily used to handle the request and then freed.

If you prefer using large, 16kB static buffers which will never need dynamic memory, you can enable this option by defining **LARGE_STATIC_BUFFERS**.

If dynamic buffers are used and the user requests an CyaSSL_write() that is bigger than the buffer size, then a dynamic block up to MAX_RECORD_SIZE is used to send the data. Users wishing to only send the data in chunks of the current buffer size at maximum, as defined by RECORD_SIZE, can do this by defining **STATIC_CHUNKS_ONLY**. When using this define, RECORD_SIZE defaults to 128 bytes.

2.5 Filesystem

Q: When do I need to read this section?
A: No file system is available, standard file system functions are not available, or you have a custom file system.

CyaSSL uses the filesystem for loading keys and certificates into the SSL session or context. CyaSSL also allows loading these from memory buffers. If strictly using memory buffers, a filesystem is not needed.

You can disable CyaSSL's usage of the filesystem by defining **NO_FILESYSTEM** when building the library. This means that certificates and keys will need to be loaded from memory buffers instead of files. An example of setting this in settings.h:

```
#ifdef MY_NEW_PLATFORM
      ...
      #define NO_FILESYSTEM
      ...
#endif
```

Test key and certificate buffers can be found in the *./cyassl/certs_test.h* header file. These will match up to corresponding certificates and keys found in the *./certs* directory.

The *certs_test.h* header file can be updated using the *./gencertbuf.pl* script if needed. Inside *gencertbuf.pl*, there are two arrays: **fileList_1024** and **fileList_2048**. Additional certificates or keys may be added to the respective array, depending on key size, and must be in DER format. The above mentioned arrays map a certificate/key file location with the desired buffer name. After modifying *gencertbuf.pl*, running it from the CyaSSL root directory will update the certificate and key buffers in *./cyassl/certs_test.h*:

```
./gencertbuf.pl
```

If you would like to use a filesystem other than the default, the filesystem abstraction layer

is located in *./src/ssl.c*. Here you will see filesystem ports for various platforms including EBSNET, FREESCALE_MQX, and MICRIUM. You can add a custom define for your platform if needed - allowing you to define file system functions with XFILE, XFOPEN, XFSEEK, etc. For example, the filesystem layer in ssl.c for Micrium's µC/OS (MICRIUM) is as follows:

```
#elif defined(MICRIUM)
      #include <fs.h>
      #define XFILE      FS_FILE*
      #define XFOPEN     fs_fopen
      #define XFSEEK     fs_fseek
      #define XFTELL     fs_ftell
      #define XREWIND    fs_rewind
      #define XFREAD     fs_fread
      #define XFCLOSE    fs_fclose
      #define XSEEK_END  FS_SEEK_END
      #define XBADFILE   NULL
```

2.6 Threading

Q: When do I need to read this section?
A: You want to use CyaSSL in a multithreaded environment, or want to just compile it in single threaded mode.

If CyaSSL will only be used in a single threaded environment, the CyaSSL mutex layer can be disabled when compiling CyaSSL by defining **SINGLE_THREADED**. This will negate the need to port the CyaSSL mutex layer.

If CyaSSL needs to be used in a multithreaded environment, the CyaSSL mutex layer will need to be ported to the new environment. The mutex layer can be found in *./cyassl/ctaocrypt/port.h* and *./ctaocrypt/src/port.c*. **CyaSSL_Mutex** will need to be defined for the new system in *port.h* and the mutex functions(InitMutex, FreeMutex, etc.) in *port.c*. Search in *port.h* and *port.c* for some existing platform port layers (EBSNET, FREESCALE_MQX, etc.) as examples.

2.7 Random Seed

Q: When do I need to read this section?
A: Either */dev/random* or */dev/urandom* is not available or you want to integrate into a hardware RNG.

By default, CyaSSL uses */dev/urandom* or */dev/random* to generate a RNG seed. The **NO_DEV_RANDOM** define can be used when building CyaSSL to disable the default GenerateSeed() function. If this is defined, you need to write a custom GenerateSeed() function in *./ctaocrypt/src/random.c*, specific to your target platform. This allows you to seed CyaSSL's PRNG with a hardware-based random entropy source if available.

For examples of how GenerateSeed() needs to be written, reference CyaSSL's existing GenerateSeed() implementations in *./ctaocrypt/src/random.c*.

2.8 Memory

Q: When do I need to read this section?
A: When you don't have standard memory functions available or are interested in memory usage differences between optional math libraries.

CyaSSL proper uses both malloc() and free() by default. When using the normal big integer math library, CTaoCrypt will also use realloc().

By default CyaSSL/CTaoCrypt use the normal big integer math library, which uses quite a bit of dynamic memory. When building CyaSSL, the fastmath library can be enabled, which is both faster and uses no dynamic memory for crypto operations (all on the stack). By using fastmath, CyaSSL won't need a realloc() implementation at all. As the SSL layer of CyaSSL still uses some dynamic memory, malloc() and free() are still required.

For a comparison of resource usage (stack/heap) between the big integer math library and fastmath library, ask us to see our Resource Use document.

To enable fastmath, define **USE_FAST_MATH** and build in *./ctaocrypt/src/tfm.c* instead of *./ctaocrypt/src/integer.c*. Since the stack memory can be large when using fastmath, we recommend defining **TFM_TIMING_RESISTANT** as well.

If the normal malloc(), free(), and possibly realloc() functions are not available, define **XMALLOC_USER**, then provide custom memory function hooks in *./cyassl/ctaocrypt/types.h* specific to the target environment.

Please read section 5.1.1.1 of the CyaSSL Manual for details about using XMALLOC_USER: http://yassl.com/yaSSL/Docs-cyassl-manual-5-portability.html

2.9 Time

Q: When do I need to read this section?
A: When standard time functions (time(), gmtime()) are not available, or you need to specify a custom clock tick function.

By default, CyaSSL uses *time()*, *gmtime()*, and *ValidateDate()*, as specified in *./ctaocrypt/src/asn.c*. These are abstracted to XTIME, XGMTIME, and XVALIDATE_DATE. If the standard time functions, and time.h, are not available, the user can define **USER_TIME**. After defining USER_TIME, the user can define their own XTIME, XGMTIME, and XVALIDATE_DATE functions.

CyaSSL uses *time(0)* by default for the clock tick function. This is located in ./src/internal.c inside of the *LowResTimer()* function.

Defining **USER_TICKS** allows the user to define their own clock tick function if time(0) is not wanted. The custom function needs second accuracy, but doesn't have to be correlated to EPOCH. See *LowResTimer()* function in *./src/internal.c* for reference.

2.10 C Standard Library

Q: When do I need to read this section?
A: When you don't have a C standard library available, or have a custom one.

CyaSSL can be built without the C standard library to provide a higher level of portability and flexibility to developers. When doing so, the user needs to map functions they wish to use instead of the C standard ones.

Section 7, above, covered memory functions. In addition to memory function abstraction, CyaSSL also abstracts string function and math functions, where the specific functions are typically abstracted to a define in the form of X<FUNC>, where <FUNC> is the name of the function being abstracted.

Please read Section 5.1 of the CyaSSL Manual for details:
http://www.yassl.com/yaSSL/Docs-cyassl-manual-5-portability.html

2.11 Logging

Q: When do I need to read this section?
A: You want to enable debug messages but don't have stderr available.

By default, CyaSSL provides debug output through stderr. In order for debug messages to be enabled, CyaSSL must be compiled with **DEBUG_CYASSL** defined, and *CyaSSL_Debugging_ON()* must be called from the application code. *CyaSSL_Debugging_OFF()* may be used by the application layer to turn off CyaSSL debug messages.

For environments which do not have stderr available, or wish to output debug messages over a different output stream or in a different format, CyaSSL allows applications to register a logging callback.

Please read Section 8.1 of the CyaSSL Manual for details:
http://www.yassl.com/yaSSL/Docs-cyassl-manual-8-debugging.html

2.12 Public Key Operations

Q: When do I need to read this section?

A: You want to use your own public key implementation with CyaSSL.

CyaSSL allows users to write their own public key callbacks which will be called when the SSL/TLS layer needs to do public key operations. The user can optionally define 6 functions:

1. ECC sign callback
2. ECC verify callback
3. RSA sign callback
4. RSA verify callback
5. RSA encrypt callback
6. RSA decrypt callback

For full details, please read Section 6.4 of the CyaSSL Manual:
http://www.yassl.com/yaSSL/Docs-cyassl-manual-6-callbacks.html

2.13 Atomic Record Layer Processing

Q: When do I need to read this section?
A: You want to do your own processing of record layers, specifically MAC/encrypt and decrypt/verify operations.

By default, CyaSSL handles record layer processing for the user using its cryptography library, wolfCrypt. CyaSSL provides Atomic Record Processing callbacks for users who wish to have more control over MAC/encrypt and decrypt/verify functionality during the SSL/TLS connection.

The user will need to define 2 functions:
1. MAC/encrypt callback function
2. Decrypt/verify callback function

For full details, please read Section 6.3 of the CyaSSL Manual:
http://www.yassl.com/yaSSL/Docs-cyassl-manual-6-callbacks.html

2.14 Features

Q: When do I need to read this section?
A: When you want to disable features.

Features can be disabled when building CyaSSL by using the appropriate defines. For a list of defines available, please refer to Chapter 2 of the CyaSSL Manual:
http://www.yassl.com/yaSSL/Docs-cyassl-manual-2-building-cyassl.html

3. Next Steps

3.1 CTaoCrypt Test Application

After getting CyaSSL proper to build on the target platform, a good next step is to port the CTaoCrypt test application. Running this application on the target system will verify that all the crypto algorithms are working correctly, using NIST test vectors.

If this step is skipped, and you instead proceed directly to establishing an SSL connection, it can be more difficult to debug problems caused by underlying crypto operations failing.

The CTaoCrypt test application is located in *./ctaocrypt/test/test.c*. If an embedded application has its own *main()* function, then **NO_MAIN_DRIVER** may be defined when compiling *./ctaocrypt/test/test.c*. This will allow the application's main() to call each cipher/algorithm test individually on its own.

If an embedded device does not have enough resources to run the entire CTaoCrypt test application, individual tests can be broken out of *test.c* and compiled individually. Please ensure that correct header files needed for the specific test case are included in the build when extracting isolated crypto tests from *test.c*.

4. Support

General support questions may be sent directly to wolfSSL either through email, support forums, or wolfSSL's Zendesk ticket tracking system.

Website: http://www.wolfssl.com
Support Email: support@wolfssl.com
Zendesk: https://wolfssl.zendesk.com
Forums: http://www.wolfssl.com/forums

wolfSSL offers several support packages as well as consulting services to help users and customers port CyaSSL to new environments.

Support Package: http://www.yassl.com/yaSSL/Support/support_tiers.php
General Inquiries: info@wolfssl.com