emSSL

Secure Sockets Library

User Guide & Reference Manual

Document: UM15001
Software Version: 2.56
Revision: 0
Date: October 10, 2018

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

Copyright notice

Trademarks

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	`support@segger.com`
Internet:	*www.segger.com*

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please report it to us and we will try to assist you as soon as possible.

Contact us for further information on topics or functions that are not yet documented.

Print date: October 10, 2018

Software	Revision	Date	By	Description
2.56	0	181010	PC	Chapter "Configuring cryptography" • Added iMX RT10xx DCP coprocessor.
2.54a	0	180704	PC	Update to latest software version.
2.54	0	180621	PC	Chapter "Configuring cryptography" • Added STM32 AES coprocessor. • Added STM32 HASH coprocessor.
2.52	0	171116	PC	Chapter "API Reference" • Added `SSL_CLIENT_ConfigMutualAuth()`. • Added `SSL_SERVER_ConfigMutualAuth()`. • Added `SSL_SIGNATURE_SIGN_Add()`. • Added `SSL_SIGNATURE_SIGN_RSA_API`. • Added `SSL_SIGNATURE_SIGN_ECDSA_API`. • Added `SSL_SUITE_RSA_WITH_CAMELLIA_128_GCM_SHA256`. • Added `SSL_SUITE_RSA_WITH_CAMELLIA_256_GCM_SHA384`. • Added `SSL_CURVE_secp192k1` elliptic curve. • Added `SSL_CURVE_secp224k1` elliptic curve. • Added `SSL_CURVE_secp256k1` elliptic curve. Chapter "Configuring emSSL" • Added "Adding public key messsage signers".
2.50	0	170823	PC	Chapter "API Reference" • Added `SSL_SUITE_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`. • Added `SSL_SUITE_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`. • Added `SSL_SUITE_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256`. • Added `SSL_CIPHER_CHACHA20_POLY1305_API` cipher. • Added `SSL_CURVE_Curve25519` elliptic curve. • Added `SSL_GetCopyrightText()`. • Added `SSL_GetVersionText()`.
2.42	0	170728	PC	Chapter "API Reference" • Added "Preprocessor symbols". • Added `SSL_SUITE_GetCipherName()`. • Added `SSL_SUITE_GetIANASuiteName()`. • Added `SSL_SUITE_GetKeyExchangeName()`. • Added `SSL_SUITE_GetMACAlgorithmName()`. • Added `SSL_SUITE_GetPKAlgorithmName()`. • Added `SSL_SUITE_QueryNull()`. • Added `SSL_SUITE_QueryPKAlgorithm()`. • Added `SSL_CURVE_Remove()`. • Added `SSL_CURVE_GetName()`. • Added `SSL_PROTOCOL_GetText()`. • Added `SSL_CURVE_brainpoolP256r1` elliptic curve. • Added `SSL_CURVE_brainpoolP384r1` elliptic curve. • Added `SSL_CURVE_brainpoolP512r1` elliptic curve. Index "Subject index" • Added. General • Minor typographical corrections.
2.40	0	170524	PC	Chapter "Exploring emSSL" • Added. Chapter "Using emSSL" • Added. Chapter "Configuring emSSL". • Added application data fragmentation. • Added session ticket length. • Added server session cache size. • Added ARIA and SEED cipher suites. Chapter "Configuring cryptography" • Completely rewritten for emCrypt. Chapter "OS Integration" • Added reference pages for `SSL_OS_*` functions.
2.30	0	160802	PC	Chapter "Configuring emSSL".

Software	Revision	Date	By	Description
				• Removed PRF configuration. Chapter “Working with emSSL”. • Added section “Installing root certificates”. Chapter “API Reference”. • Functions are now documented in groups. • Added section “API Evolution”.
2.20	0	160530	PC	Chapter “Configuring emSSL”. • Added CRYPTO configuration.
2.12	0	160316	PC	Chapter “Configuring emSSL”. • Added Camellia cipher suites.
2.11	0	151201	PC	Chapter “Configuring emSSL”. • Added CCM and CCM-8 cipher suites.
2.10	0	150710	PC	Documentation converted to new format. • Minor documentation fixes. • Documentation in manual automatically derived from source code. • Performance tuning and memory reduction for AES.
1.02	0	150521	PC	Initial release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *C: A Reference Manual* by Harbison and Steele (ISBN 0--13--089592X). This book provides a complete description of the C language, the run-time libraries, and a style of C programming that emphasizes correctness, portability, and maintainability.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

Chapter 1

Introduction to emSSL

This section presents an overview of emSSL, its structure, and its capabilities.

1.1 What is emSSL?

emSSL is a software library that enables you to create secure connections between a client and a server, typically over the Internet using TCP/IP.

In this manual we use the term SSL to indicate a protocol supported by emSSL. SSL is the original acronym for Secure Sockets Layer, which is now more accurately known as Transport layer Security or TLS. It may seem confusing to use the old acronym SSL when talking of TLS, but SSL is so well established that the term endures in literature and product names alike: emSSL is no different.

Although SSL is usually associated with secure connections to a website using TCP/IP, the SSL specification makes no mention of TCP/IP. In fact, you can use emSSL to run an SSL session over any bidirectional channel, for instance a serial line or wireless link, and provide a secure connection.

emSSL is both hardware independent and transport independent, and integrates seamlessly with embOS/IP. For interoperability, emSSL has support for TLS versions 1.0, 1.1, and 1.2 with mandatory and extended cipher suites. Support for SSL 2 and SSL 3 is absent as these protocols are now proven insecure.

1.2 Design goals

emSSL is designed with the following goals in mind:

- Highly modular such that unused features are never linked into the application.
- Be completely runtime configurable, adding each modular feature as needed.
- Present a simple user-level API that is easy to use without extensive setup.
- Easy to maintain both by SEGGER and anybody with access to the sources.
- Conform to all necessary standards and current best practices.
- Be efficient both in terms of resource usage and execution speed.
- Target 8-bit to 32-bit processors with limited resources as well as workstations.
- Provide easily-substituted cryptography for customers that require it.

We believe all design goals are achieved by emSSL.

1.3 Features

emSSL is written in ANSI C and can be used on virtually any CPU. Here is a list of emSSL features:

- ISO/ANSI C source code.
- High performance.
- Small footprint.
- Runs “out-of-the-box”.
- Highly compact implementation runs effortlessly on single-chip MCUs.
- Standard support for TLS versions 1.0, 1.1, and 1.2.
- Easy-to-understand and simple-to-use API.
- Simple configuration.
- Secure any open channel (e.g. serial line or wireless link).
- Wide range of pluggable cipher suites for interoperability with popular servers.
- Modular architecture links only what you need.
- Plug-in hardware acceleration.
- Leading-edge cipher suites for confidentiality, integrity and authentication.
- Diffie-Hellman Ephemeral cipher suites for forward secrecy.
- Elliptic curve cipher suites for reduced certificate and key sizes.
- Royalty-free.

1.4 Package content

emSSL is provided in source code and contains everything required. The following table shows the content of the emSSL Package:

Files	Description
`Application`	emSSL sample applications for bare metal and embOS.
`Config`	Configuration header files.
`CRYPTO`	Shared cryptographic library source code.
`Doc`	emSSL documentation.
`Sample/Config`	Example emSSL user configuration.
`SEGGER`	SEGGER software component source code used in emSSL.
`SSL`	emSSL implementation source code.
`Windows/SSL`	emSSL sample applications for Windows.

1.4.1 Include directories

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- `Config`
- `CRYPTO`
- `SEGGER`
- `SSL`

Always make sure that you have only one version of each file.

Note

It is frequently a major problem when updating to a new version of emSSL if you have old files included and therefore mix different versions. If you keep emSSL in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the emSSL directories before updating.

Chapter 2

Exploring emSSL

This chapter describes how to try out emSSL on a PC and embedded hardware with minimal effort. We highly recommend that you try out a working version of emSSL, shipped by SEGGER, with a known-good setup, preferably on an emPower board, before attempting to add it to your own application.

2.1 Using a PC to try emSSL

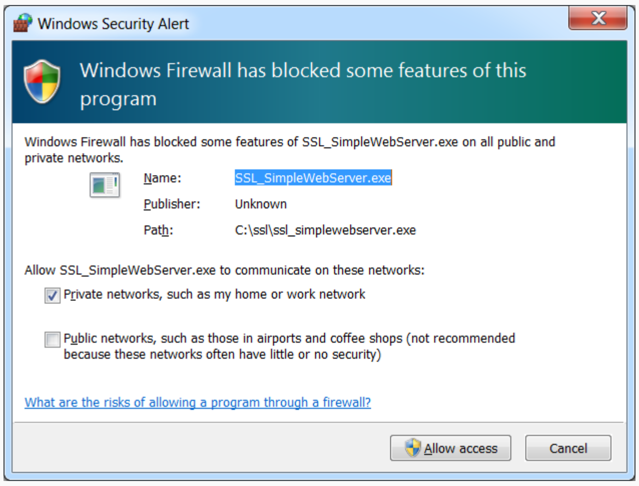
emSSL is shipped with a precompiled example that demonstrate a simple SSL web server. You can run the example and connect to the local web server on port 443.

```
C:> SSL_SimpleWebServer.exe

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL Simple Secure Web Server V2.50 compiled May 25 2017 16:22:36

Waiting for connection on port 443...
_
```

When you run this, Windows Firewall will present a dialog asking whether to grant network access to the application:



Windows Security Alert

Windows Firewall has blocked some features of this program

Windows Firewall has blocked some features of SSL_SimpleWebServer.exe on all public and private networks.

Name: SSL_SimpleWebServer.exe
Publisher: Unknown
Path: C:\ssl\ssl_simplewebserver.exe

Allow SSL_SimpleWebServer.exe to communicate on these networks:

[✓] Private networks, such as my home or work network

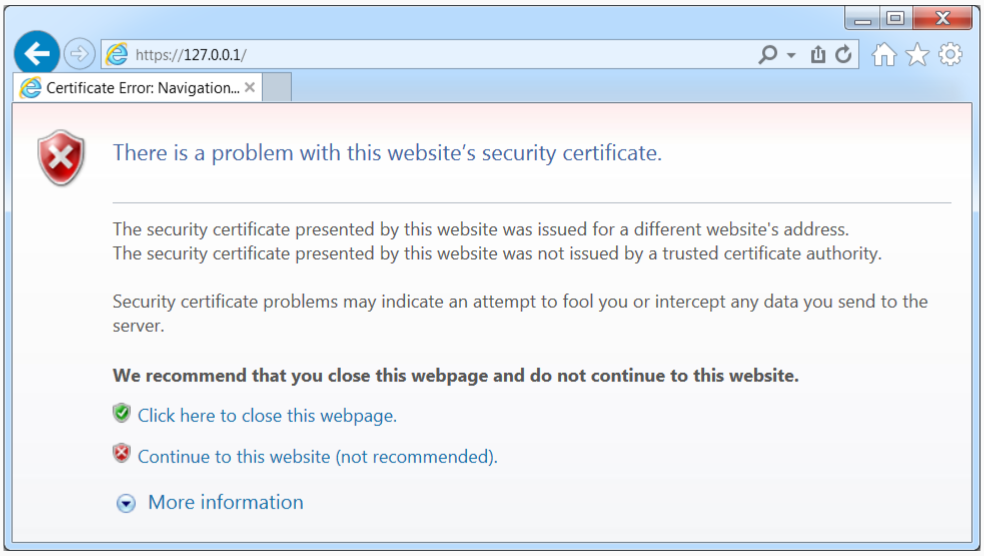
[ ] Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing a program through a firewall?

Windows Firewall dialog

Proceed and grant access otherwise you will not be able to serve web pages to clients.

The web server application is waiting for a client to connect to it such that it can serve its small web page. Now start a web browser and open the URL “`https://127.0.0.1/`”. This example uses Internet Explorer 11:



https://127.0.0.1/

Certificate Error: Navigation...

There is a problem with this website’s security certificate.

The security certificate presented by this website was issued for a different website's address.
The security certificate presented by this website was not issued by a trusted certificate authority.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

Continue to this website (not recommended).

More information

Windows Firewall dialog

The warning shown by the browser indicates that the certificate presented is invalid — and it is, according to the browser, because you are browsing your own PC using a self-signed certificate rather than a fully-authenticated certificate for a website on the Internet.

You will notice that the server has accepted the connection, negotiated the connection, and then closed the connection and is waiting for a new connection:

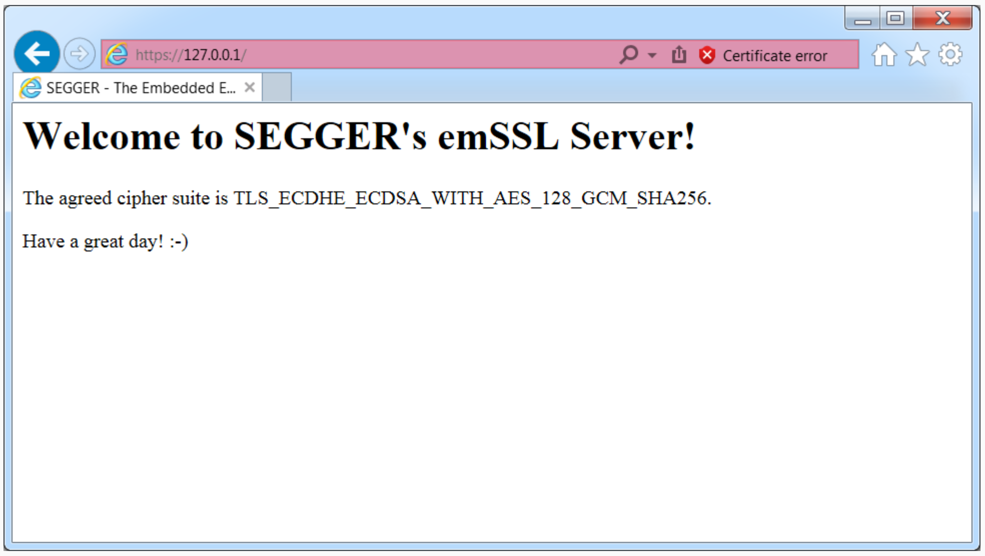
```
C:> SSL_SimpleWebServer.exe

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL Simple Secure Web Server V2.40 compiled May 25 2017 16:22:36

Waiting for connection on port 443...
Connection made, attempting to upgrade to secure...
Session is now secured by TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Session closed.

Waiting for connection on port 443...
_
```

In Internet Explorer, click "Continue to this website" or, if you are using another browser, accept the certificate or click "Advanced" and "Proceed to 127.0.0.1" and you should be greeted with a short web page served by emSSL on your PC:



Welcome to SEGGER's emSSL Server!

The agreed cipher suite is TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.

Have a great day! :-)

Page served by emSSL

The browser makes two additional connections to the server to gather the web page and any favicon. Refeshing the page in the browser will cause only one secure connection to be made to the server.

Type **Ctrl+C** to close the emSSL web server:

```
C:> SSL_SimpleWebServer.exe

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL Simple Secure Web Server V2.40 compiled May 25 2017 16:22:36

Waiting for connection on port 443...
Connection made, attempting to upgrade to secure...
Session is now secured by TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Session closed.

Waiting for connection on port 443...
Connection made, attempting to upgrade to secure...
Session is now secured by TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Session closed.

Waiting for connection on port 443...
Connection made, attempting to upgrade to secure...
Session is now secured by TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Socket closed by server.
```

```
Waiting for connection on port 443...
^C
C:> _
```

This shows that both sides of the TLS connection are working correctly and the cipher suite that was agreed between them is `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`.

2.2 Scanning websites with emSSL

The previous section demonstrated emSSL running in server mode. emSSL is shipped with a precompiled example that demonstrates client mode.

Open a command line window and navigate to the `Windows/SSL` directory that contains the `SSL_Scan.exe` application. Once there, run `SSL_Scan.exe` on `www.segger.com` and you should see something similar to this:

```
C:> ssl_scan www.segger.com

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL TLS Scan V2.40 compiled May 25 2017 17:22:32

Scanning cipher suites for www.segger.com:443...

0084 RSA_WITH_CAMELLIA_256_CBC_SHA             TLS 1.2  RSA      72 ms
0041 RSA_WITH_CAMELLIA_128_CBC_SHA             TLS 1.2  RSA      73 ms
009D RSA_WITH_AES_256_GCM_SHA384               TLS 1.2  RSA      74 ms
003D RSA_WITH_AES_256_CBC_SHA256               TLS 1.2  RSA      73 ms
0035 RSA_WITH_AES_256_CBC_SHA                  TLS 1.2  RSA      74 ms
009C RSA_WITH_AES_128_GCM_SHA256               TLS 1.2  RSA      74 ms
003C RSA_WITH_AES_128_CBC_SHA256               TLS 1.2  RSA      71 ms
002F RSA_WITH_AES_128_CBC_SHA                  TLS 1.2  RSA      73 ms
000A RSA_WITH_3DES_EDE_CBC_SHA                 TLS 1.2  RSA      74 ms
C030 ECDHE_RSA_WITH_AES_256_GCM_SHA384         TLS 1.2  RSA      75 ms
C028 ECDHE_RSA_WITH_AES_256_CBC_SHA384         TLS 1.2  RSA      80 ms
C014 ECDHE_RSA_WITH_AES_256_CBC_SHA            TLS 1.2  RSA      79 ms
C02F ECDHE_RSA_WITH_AES_128_GCM_SHA256         TLS 1.2  RSA      78 ms
C027 ECDHE_RSA_WITH_AES_128_CBC_SHA256         TLS 1.2  RSA      76 ms
C013 ECDHE_RSA_WITH_AES_128_CBC_SHA            TLS 1.2  RSA      78 ms
C012 ECDHE_RSA_WITH_3DES_EDE_CBC_SHA           TLS 1.2  RSA      77 ms
0088 DHE_RSA_WITH_CAMELLIA_256_CBC_SHA         TLS 1.2  RSA     179 ms
0045 DHE_RSA_WITH_CAMELLIA_128_CBC_SHA         TLS 1.2  RSA     177 ms
009F DHE_RSA_WITH_AES_256_GCM_SHA384           TLS 1.2  RSA     176 ms
006B DHE_RSA_WITH_AES_256_CBC_SHA256           TLS 1.2  RSA     174 ms
0039 DHE_RSA_WITH_AES_256_CBC_SHA              TLS 1.2  RSA     177 ms
009E DHE_RSA_WITH_AES_128_GCM_SHA256           TLS 1.2  RSA     178 ms
0067 DHE_RSA_WITH_AES_128_CBC_SHA256           TLS 1.2  RSA     175 ms
0033 DHE_RSA_WITH_AES_128_CBC_SHA              TLS 1.2  RSA     179 ms
0016 DHE_RSA_WITH_3DES_EDE_CBC_SHA             TLS 1.2  RSA     180 ms

25 common cipher suites out of 106 tested

C:> _
```

This shows that emSSL has made 25 successful connections to `www.segger.com` out of 106 different protocols attempted. Reading the columns from left to right we see:

- The hexadecimal ID of the agreed cipher suite.
- The IANA name for the agreed cipher suite (without the “`TLS_`” prefix).
- The TLS protocol version that was agreed between emSSL and the host.
- The type of public key used for key exchange.
- How long the connection took to set up and agree.

For a different set of cipher suites, scan Facebook:

```
C:> scan www.facebook.com

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL TLS Scan V2.40 compiled May 25 2017 17:22:32

Scanning cipher suites for www.facebook.com:443...

0005 RSA_WITH_RC4_128_SHA                      TLS 1.2  RSA      68 ms
009D RSA_WITH_AES_256_GCM_SHA384               TLS 1.2  RSA      81 ms
0035 RSA_WITH_AES_256_CBC_SHA                  TLS 1.2  RSA      66 ms
```

```
009C RSA_WITH_AES_128_GCM_SHA256              TLS 1.2  RSA      72 ms
002F RSA_WITH_AES_128_CBC_SHA                 TLS 1.2  RSA      68 ms
000A RSA_WITH_3DES_EDE_CBC_SHA                TLS 1.2  RSA      69 ms
C011 ECDHE_RSA_WITH_RC4_128_SHA               TLS 1.2  RSA      76 ms
C030 ECDHE_RSA_WITH_AES_256_GCM_SHA384        TLS 1.2  RSA      73 ms
C014 ECDHE_RSA_WITH_AES_256_CBC_SHA           TLS 1.2  RSA      72 ms
C02F ECDHE_RSA_WITH_AES_128_GCM_SHA256        TLS 1.2  RSA      75 ms
C013 ECDHE_RSA_WITH_AES_128_CBC_SHA           TLS 1.2  RSA      78 ms
C012 ECDHE_RSA_WITH_3DES_EDE_CBC_SHA          TLS 1.2  RSA      73 ms
C007 ECDHE_ECDSA_WITH_RC4_128_SHA             TLS 1.2  ECDSA    94 ms
C02C ECDHE_ECDSA_WITH_AES_256_GCM_SHA384      TLS 1.2  ECDSA    84 ms
C00A ECDHE_ECDSA_WITH_AES_256_CBC_SHA         TLS 1.2  ECDSA    80 ms
C02B ECDHE_ECDSA_WITH_AES_128_GCM_SHA256      TLS 1.2  ECDSA    83 ms
C009 ECDHE_ECDSA_WITH_AES_128_CBC_SHA         TLS 1.2  ECDSA    80 ms
C008 ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA        TLS 1.2  ECDSA    84 ms

18 common cipher suites out of 106 tested

C:> _
```

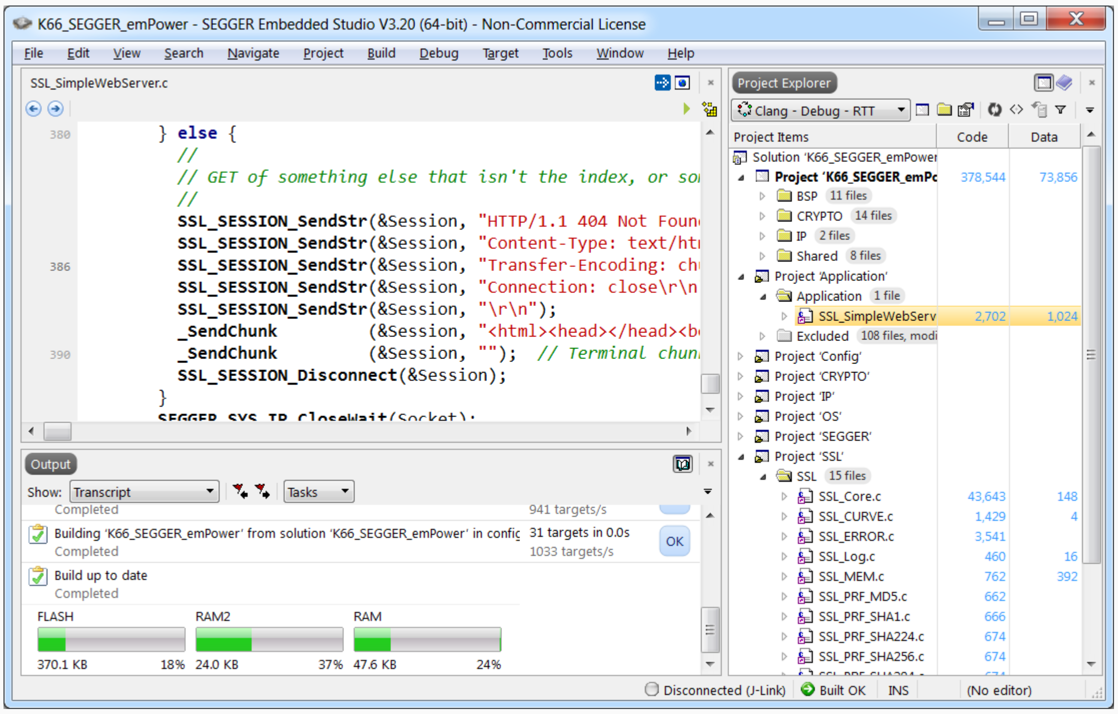
Many of the cipher suites are the same as before, with new ECDHE-ECDSA suites appearing. In fact, most servers on the Internet use only a small subset of the vast range of cipher suites, key sizes, and elliptic curves that are available to be negotiated.

2.3 Moving to embedded hardware

When starting to run emSSL on embedded hardware, we recommend that you use one of the “Start” projects suppied in the BSP for your target system to begin with and gain confidence with a working system before progressing to add emSSL to your own application.

The following sections describe this process using SEGGER Embedded Studio, but the principles are the same for any embedded development or workstation environment. The target hardware is an SEGGER emPower board which is supplied with Embedded Studio PRO or available separately from SEGGER and through authorized distributors.

Start Embedded Studio and load the SEGGER emPower start project:


K66_SEGGER_emPower - SEGGER Embedded Studio V3.20 (64-bit) - Non-Commercial License
File Edit View Search Navigate Project Build Debug Target Tools Window Help
SSL_SimpleWebServer.c
Project Explorer
Clang - Debug - RTT
Output
Show: Transcript
Tasks
OK
Disconnected (J-Link)
Built OK
INS
(No editor)

```
        } else {
          //
          // GET of something else that isn't the index, or so
          //
          SSL_SESSION_SendStr(&Session, "HTTP/1.1 404 Not Foun
          SSL_SESSION_SendStr(&Session, "Content-Type: text/ht
          SSL_SESSION_SendStr(&Session, "Transfer-Encoding: ch
          SSL_SESSION_SendStr(&Session, "Connection: close\r\n
          SSL_SESSION_SendStr(&Session, "\r\n");
          _SendChunk         (&Session, "<html><head></head><b
          _SendChunk         (&Session, "");  // Terminal chun
          SSL_SESSION_Disconnect(&Session);
        }
```

Project Items	Code	Data
Solution 'K66_SEGGER_emPower		
Project 'K66_SEGGER_emPo	378,544	73,856
BSP 11 files		
CRYPTO 14 files		
IP 2 files		
Shared 8 files		
Project 'Application'		
Application 1 file		
SSL_SimpleWebServ	2,702	1,024
Excluded 108 files, modi		
Project 'Config'		
Project 'CRYPTO'		
Project 'IP'		
Project 'OS'		
Project 'SEGGER'		
Project 'SSL'		
SSL 15 files		
SSL_Core.c	43,643	148
SSL_CURVE.c	1,429	4
SSL_ERROR.c	3,541	
SSL_Log.c	460	16
SSL_MEM.c	762	392
SSL_PRF_MD5.c	662	
SSL_PRF_SHA1.c	666	
SSL_PRF_SHA224.c	674	
SSL_PRF_SHA256.c	674	

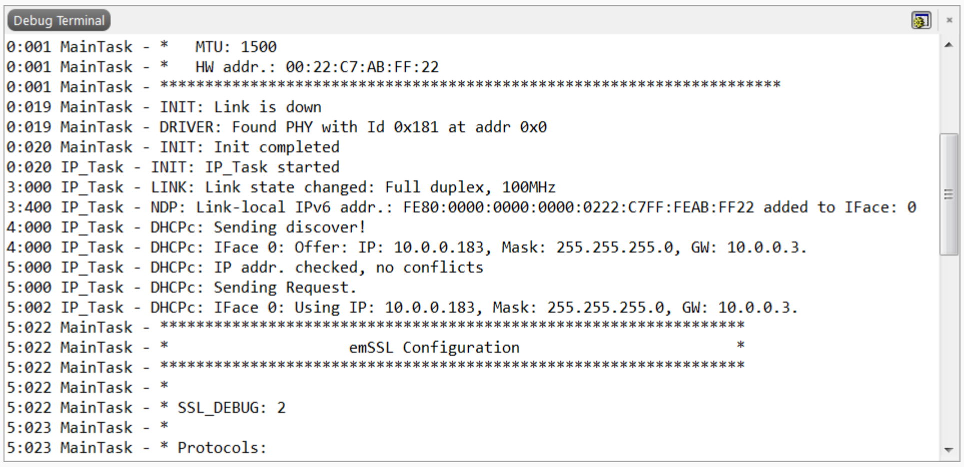
```
Completed                                                941 targets/s
Building 'K66_SEGGER_emPower' from solution 'K66_SEGGER_emPower' in config   31 targets in 0.0s
Completed                                                1033 targets/s
Build up to date
Completed
```

FLASH	RAM2	RAM
370.1 KB 18%	24.0 KB 37%	47.6 KB 24%

Start project in Embedded Studio

Once you have loaded your start project, you can test it out by choosing **Debug > Go** which flashes it into your target and starts running it under control of the debugger.

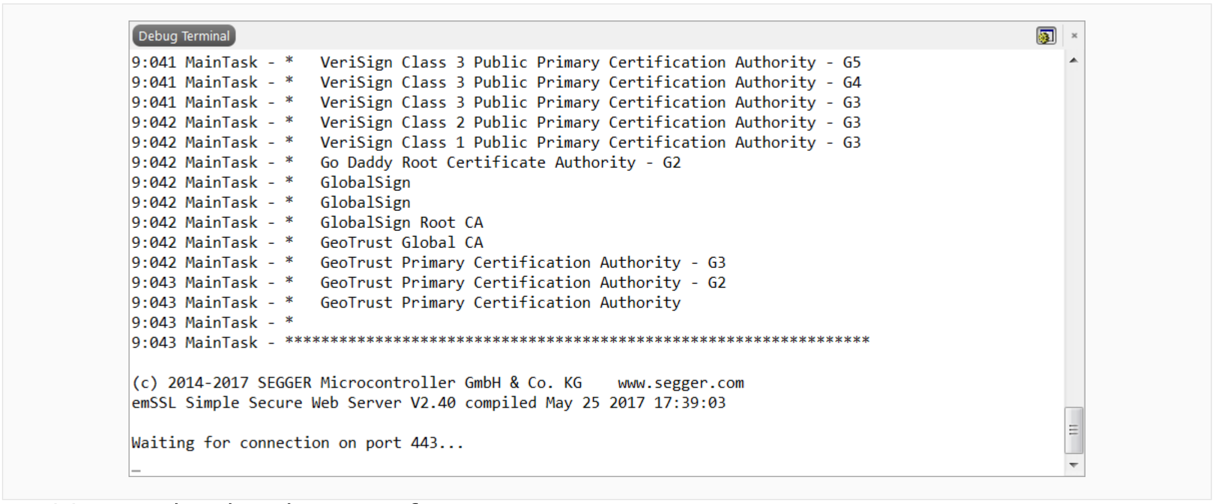
The **Debug Terminal** will show the configured IP address of the emPower board and you will be able to use a browser to show web pages serverd from the embedded target in the same way as the PC application above. If you do not see the **Debug Terminal**, choose **View > Debug Terminal**. The terminal output will look something similar to this:



```
0:001 MainTask - *   MTU: 1500
0:001 MainTask - *   HW addr.: 00:22:C7:AB:FF:22
0:001 MainTask - *********************************************************************
0:019 MainTask - INIT: Link is down
0:019 MainTask - DRIVER: Found PHY with Id 0x181 at addr 0x0
0:020 MainTask - INIT: Init completed
0:020 IP_Task - INIT: IP_Task started
3:000 IP_Task - LINK: Link state changed: Full duplex, 100MHz
3:400 IP_Task - NDP: Link-local IPv6 addr.: FE80:0000:0000:0000:0222:C7FF:FEAB:FF22 added to IFace: 0
4:000 IP_Task - DHCPc: Sending discover!
4:000 IP_Task - DHCPc: IFace 0: Offer: IP: 10.0.0.183, Mask: 255.255.255.0, GW: 10.0.0.3.
5:000 IP_Task - DHCPc: IP addr. checked, no conflicts
5:000 IP_Task - DHCPc: Sending Request.
5:002 IP_Task - DHCPc: IFace 0: Using IP: 10.0.0.183, Mask: 255.255.255.0, GW: 10.0.0.3.
5:022 MainTask - *****************************************************************
5:022 MainTask - *                    emSSL Configuration                        *
5:022 MainTask - *****************************************************************
5:022 MainTask - *
5:022 MainTask - * SSL_DEBUG: 2
5:023 MainTask - *
5:023 MainTask - * Protocols:
```

Log output in Debug Terminal

emSSL will then display its configuration and indicate that it’s waiting for a connection:



```
9:041 MainTask - *   VeriSign Class 3 Public Primary Certification Authority - G5
9:041 MainTask - *   VeriSign Class 3 Public Primary Certification Authority - G4
9:041 MainTask - *   VeriSign Class 3 Public Primary Certification Authority - G3
9:042 MainTask - *   VeriSign Class 2 Public Primary Certification Authority - G3
9:042 MainTask - *   VeriSign Class 1 Public Primary Certification Authority - G3
9:042 MainTask - *   Go Daddy Root Certificate Authority - G2
9:042 MainTask - *   GlobalSign
9:042 MainTask - *   GlobalSign
9:042 MainTask - *   GlobalSign Root CA
9:042 MainTask - *   GeoTrust Global CA
9:042 MainTask - *   GeoTrust Primary Certification Authority - G3
9:043 MainTask - *   GeoTrust Primary Certification Authority - G2
9:043 MainTask - *   GeoTrust Primary Certification Authority
9:043 MainTask - *
9:043 MainTask - ******************************************************************

(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL Simple Secure Web Server V2.40 compiled May 25 2017 17:39:03

Waiting for connection on port 443...
_
```

emSSL initialized and waiting for a connection

At this point you will be able to use Internet Explorer, or your favorite web browser, to view secure content served by the emPower board.

Chapter 3

Using emSSL

This chapter presents a simple secure client and server that demonstrates how to integrate emSSL into your application.

In this section we assume that you will use a PC or have a fully-functioning embOS/IP project that is able to connect to the network and all that is required is to add emSSL to the project.

3.1 Sample applications

emSSL ships with a number of sample applications that demonstrate how to integrate SSL into your application.

The sample applications are:

Application	Description
`SSL_ROT13Server.c`	A server that provides a ROT13 service.
`SSL_ROT13Client.c`	A client that uses the ROT13 service.
`SSL_SimpleWebServer.c`	A minimal web server.
`SSL_SimpleWebClient.c`	A client that retrieves web content.

In this section we will describe only the ROT13 client and server.

3.1.1 A note on the samples

Each sample that presented in this section is written in a style that makes it easy to describe and that fits comfortably within the margins of printed paper. Therefore, it may well be that you would rewrite the sample to have a slightly different structure that fits better, but please keep in mind that these examples are written with clarity as the prime objective, and to that end we sacrifice some brevity and efficiency.

3.1.2 Where to find the sample code

All samples are included in the `Application` directory of the emSSL distribution.

3.2 What to expect

The following sections describe a client-server pair of applications that provode a "secure ROT13 service." The ROT13 server accepts lines of text from a client, applies the ROT13 transform to each, and sends back the results to the client. For details of ROT13, see https://en.wikipedia.org/wiki/ROT13.

Run the server

Precompiled Windows executables for both client and server are provided in the `Application` folder. Open a command line window and run the `SSL_ROT13Server` application:

```
C:> SSL_ROT13Server.exe

(c) 2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL ROT13 Server compiled May 26 2017 15:24:26


_
```

At this point the server is waiting for a connection from a client.

Run the client

Open a second command line window and run the `SSL_ROT13Client` application:

```
C:> SSL_ROT13Client.exe

(c) 2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL ROT13 Client compiled May 26 2017 15:24:26

Sent: SEGGER - The Embedded Experts
Recv: FRTTRE - Gur Rzorqqrq Rkcregf
Sent: FRTTRE - Gur Rzorqqrq Rkcregf
Recv: SEGGER - The Embedded Experts
Sent: SEGGER - It simply works!
Recv: FRTTRE - Vg fvzcyl jbexf!
Sent: FRTTRE - Vg fvzcyl jbexf!
Recv: SEGGER - It simply works!

C:> _
```

What's happening?

From the client's perspective it:

- Connects to the ROT13 server
- Sends the text "SEGGER - The Embedded Experts" to the server
- Receives the response "FRTTRE - Gur Rzorqqrq Rkcregf" from the server
- Sends the text "FRTTRE - Gur Rzorqqrq Rkcregf" to the server
- Receives the response "SEGGER - The Embedded Experts" from the server
- Does the same for "SEGGER - It simply works!"
- Closes the connection

This shows that two successive applications of ROT13 restore the original text, so the ROT13 server can both "encipher" and "decipher" using ROT13.

The server also traces what it is doing:

```
C:> SSL_ROT13Server.exe

(c) 2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL ROT13 Server compiled May 26 2017 15:24:26

Recv: SEGGER - The Embedded Experts!
Sent: FRTTRE - Gur Rzorqqrq Rkcregf!
Recv: FRTTRE - Gur Rzorqqrq Rkcregf!
```

```
Sent: SEGGER - The Embedded Experts!
Recv: SEGGER - It simply works!
Sent: FRTTRE - Vg fvzcyl jbexf!
Recv: FRTTRE - Vg fvzcyl jbexf!
Sent: SEGGER - It simply works!
_
```

From the server's perspective it:

- Accepts a connection from a client
- Receives the text "SEGGER - The Embedded Experts" from the client
- Send the response "FRTTRE - Gur Rzorqqrq Rkcregf" to the client
- Receives the text "FRTTRE - Gur Rzorqqrq Rkcregf" from the client
- Send the response "SEGGER - The Embedded Experts" to the client
- Does the same for "SEGGER - It simply works!"
- Sees that client drops the connection
- Waits for another connection

Although the connection is dropped by the client, the server does not exit, it continues executing awaiting another connection. To close the server, type **Ctrl+C**.

3.3 ROT13 server

The first application, `SSL_ROT13Server.c`, provides a service that will transform lines of text to their ROT13-encoded equivalent.

For a complete listing of this application, see *SSL_ROT13Server.c complete listing* on page 33.

3.3.1 Application entry

The main application task is responsible for setting up the environment ready to accept incoming SSL requests. This is simply boilerplate code that has no configuration:

```
void MainTask(void) {
  SSL_SESSION Session;
  int         BoundSocket;
  int         Socket;
  int         Status;
  //
  SEGGER_SYS_Init();  ❶
  SEGGER_SYS_IP_Init();
  SSL_Init();  ❷
  //
  SEGGER_SYS_IO_Printf("\n");  ❸
  SEGGER_SYS_IO_Printf("(c) 2017 SEGGER Microcontroller GmbH & Co. KG"
                       "    www.segger.com\n");
  SEGGER_SYS_IO_Printf("emSSL ROT13 Server ");
  SEGGER_SYS_IO_Printf("compiled " __DATE__ " " __TIME__ "\n\n");
  //
```

❶ **Initialize system components**

The calls to `SEGGER_SYS_Init()` and `SEGGER_SYS_IP_Init()` use the SEGGER system abstraction layer to initialize services to the application.

How you open a socket to the remote server depends on the underlying networking API. To make connection as simple as possible, emSSL examples use a common API and emSSL ships with example implementations of the API for both Windows and embOS/IP.

The emSSL examples can run on a standard Windows or Linux host, or an embedded target using embOS/IP. All SEGGER portability wrapper preprocessor symbols and functions are prefixed with “`SEGGER_SYS_`”.

❷ **Initialize SSL component**

Before using any emSSL service you must initialize the SSL module. You do this by including the emSSL header `SSL.h` and by calling `SSL_Init()`.

Configuration of the emSSL’s capabilities is carried out by `SSL_X_Config()` that is called as part of the SSL initialization carried out by `SSL_Init()`. `SSL_X_Config()` must be provided in your application as a function with external linkage and an example is shipped with emSSL. Specific configuration capabilities are not discussed in detail here, you can find extensive documentation on how to configure emSSL in *Configuring emSSL* on page 130.

❸ **Display identification**

When running the server, this code just shows that the server is up and ready for connections.

3.3.2 Accepting SSL connections

Once emSSL is correctly configured, the application in responsible for accepting connections:

```
//
// Bind application's ROT13 port.
//
BoundSocket = SEGGER_SYS_IP_Bind(19000);  ❶
if (BoundSocket < 0) {
  SEGGER_SYS_OS_Halt(100);
}
//
for (;;) {
  //
  do {  ❷
    Socket = SEGGER_SYS_IP_Accept(BoundSocket);
  } while (Socket < 0);
  //
  SSL_SESSION_Prepare(&Session, Socket, &_IP_Transport);  ❸
  Status = SSL_SESSION_Accept(&Session);  ❹
  //
  if (Status < 0) {
    SEGGER_SYS_IO_Printf("Can't negotiate a secure connection.\n\n");
    SEGGER_SYS_IP_Close(Socket);
  } else {
    do {
      Status = _Serve(&Session);  ❺
    } while (Status >= 0);
    SSL_SESSION_Disconnect(&Session);  ❻
    SEGGER_SYS_IP_CloseWait(Socket);
  }
}
```

❶ Bind the application port

There is no standard “secure ROT13 service” port, so application port 19000 is dedicated to the service.

The call to `SEGGER_SYS_IP_Bind()` uses the SEGGER abstraction layer to bind port 19000 and return a socket corresponding to that binding. If the port is already bound and cannot accept incoming connections, the application terminates.

❷ Accept an incoming connection

Once the port is bound, we listen for incoming connections. The call to `SEGGER_SYS_IP_Accept()` waits for an incoming connection and creates a socket for that connection.

❸ Prepare the SSL session for the connection

The call to `SSL_SESSION_Prepare()` initializes an SSL session. For this simple example we only deal with a single session and therefore the session is allocated in the stack frame.

`SSL_SESSION_Prepare()` is provided a set of function pointers, in a structure, that vector to the appropriate send and receive functions for a socket. In this example, we use the SEGGER abstraction layer to provide socket services:

```
static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  NULL
};
```

These are very thin “shims” to the underlying embOS/IP or Windows socket functions, with the shim providing a consistent function prototype that adapts between the various implementations available.

Although SSL is typically used over TCP/IP, it is not necessarily the only medium for SSL communications. For example, CANopen specifies a “shell” port that runs a user-defined protocol that could, quite literally, be secured by SSL. In this case, your application could use emSSL to service TCP/IP connections *and* CAN connections using the same code but with different transport APIs: one for TCP/IP and one for CAN. And, if you wish to secure a serial connection, you could add functions that read and write over (one or more) serial connections.

❹ Set up secure connection

`SSL_SESSION_Accept()` attempts to negotiate a secure session between client and server using the socket. If a the client and server can agree a common set of communication parameters, the secure connection is established and the server continues; if not, the connection is dropped and the socket is closed to terminate communication.

❺ Run the server

When the secure communication channel is successfully negotiated, processing is handed off to code that reads requests and writes responses one request at a time. This code is presented below. When the session is closed, the server exits the loop.

❻ Disconnect and close down

The session may close gracefully or abruptly, and when closed control returns from `_Serve()`. At this point, the session is disconnected (if it not already disconnected) and is fully closed from an API perspective, and no further calls should be made to the API using that closed session: doing so leads to undefined behavior.

Once the session is disconnected, the socket is closed.

3.3.3 Serving the connection

In the previous section, serving the established connection delegates to the function `_Serve()`:

```
static void _Serve(SSL_SESSION *pSession) {  ❶
  char aData[256];
  int  Status;
  //
  Status = _RdLine(pSession, aData, sizeof(aData));  ❷
  if (Status >= 0) {
    SEGGER_SYS_IO_Printf("Recv: %s", aData);
    _ApplyROT13(aData, Status);  ❸
    Status = SSL_SESSION_Send(pSession, &aData[0], Status);  ❹
    if (Status >= 0) {
      SEGGER_SYS_IO_Printf("Sent: %s", aData);
    } else {
      SEGGER_SYS_IO_Printf("Error sending data: %s\n",
                           SSL_ERROR_GetText(Status));
    }
  } else {
    if (Status != SSL_ERROR_EOF) {  ❺
      SEGGER_SYS_IO_Printf("Error receiving data: %s\n",
                           SSL_ERROR_GetText(Status));
    }
  }
  //
  return Status;  ❻
}
```

❶ Receive parameters

The function is passed a pointer to the SSL session to serve.

❷ Read an incoming line

The process of reading a single line is delegated to a function. The incoming line is deposited into aData which is zero-terminated by _RdLine(). The value returned by _RdLine() indicates the number of characters read successfully and that the connection remains open, or whether there was an error on the connection and the connection is errored.

Note that the number of characters is stored in Status to be used later, the same with the error status.

❸ Apply ROT13 tranform

If the line is received without error, it's transformed in-place using _ApplyROT13().

❹ Send response

Once transformed, the line is sent back to the client. The number of characters in the line is the result of _RdLine() which is stored in the Status variable.

❺ Handling errors

If reading the line results in an error, a diagnostic is printed. The status code SSL_ERROR_EOF is distinguished and indicates that the connection has been closed by the client end and is not reported as an error.

❻ Exit

Once processing is complete, the status of reading, transforming, and sending the line to the client is returned to the caller.

3.3.4 Reading a line of text

The client sends the server a line of text and terminates it by a newline character, '\n'. Therefore, it is not known in advance how many characters there are in the line. The function _RdLine() must therefore read one character at a time and search for the newline:

```
static int _RdLine(SSL_SESSION *pSession, U8 *pData, unsigned DataLen) {
  unsigned Len;
  int      Status;
  U8       Char;
  //
  Len = 0;
  for (;;) {
    Status = SSL_SESSION_Receive(pSession, &Char, 1);  ❶
    if (Status == 0) {  ❷
      return SSL_ERROR_EOF;
    } else if (Status < 0) {  ❸
      return Status;
    }
    pData[Len] = Char;  ❹
    if (Len+1 < DataLen) {
      ++Len;
    }
    if (Char == '\n') {  ❺
      pData[Len] = 0;
      return Len;
    }
  }
}
```

❶ Read a character

The function SSL_SESSION_Receive() reads data from the secure socket. In this case we pass in the session and provide a single-character buffer that we wish to fill.

❷ **Deal with premature socket closure**

The value returned from SSL_SESSION_Receive() indicates the status of the read. If the value is zero, the socket was gracefully closed without delivering any requested data—and in this case, because we have not received the newline, this is an unexpected state. When this happens, we elect to deliver an “end of file” error to the caller.

❸ **Propagate protocol errors**

If the value returned from SSL_SESSION_Receive() is negative, it indicates a protocol error. On receiving an error indication, _RdLine() propagates the error to the caller.

❹ **Accumulate characters**

Having dealt with premature socket closure and protocol errors, processing continues and the received character is added to the provided buffer ensuring there is enough space to hold the character and a required zero terminator.

❺ **Finalize**

Once the newline is found, the provided buffer is zero-terminated and the number of bytes deposited into the buffer, including the newline but excluding the zero terminator, is returned.

3.3.5 Transforming characters using ROT13

Now that the server part is covered, all that remains is to show the mechanics of the ROT13 transform. The code is not described further.

```
static void _ApplyROT13(U8 *pData, unsigned DataLen) {
  unsigned i;
  //
  for (i = 0; i < DataLen; ++i) {
    if ('a' <= pData[i] && pData[i] <= 'm') {
      pData[i] = pData[i] - 'a' + 'n';
    } else if ('n' <= pData[i] && pData[i] <= 'z') {
      pData[i] = pData[i] - 'n' + 'a';
    } else if ('A' <= pData[i] && pData[i] <= 'M') {
      pData[i] = pData[i] - 'A' + 'N';
    } else if ('N' <= pData[i] && pData[i] <= 'Z') {
      pData[i] = pData[i] - 'N' + 'A';
    }
  }
}
```

3.3.6 SSL_ROT13Server.c complete listing

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : SSL_ROT13Server.c
Purpose     : Simple server that provides a secure ROT13 service.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSL.h"
#include "SEGGER_SYS.h"
#include <stdio.h>
#include <stdlib.h>

/*********************************************************************
*
*             Static const data
*
**********************************************************************
*/

static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  NULL
};

/*********************************************************************
*
*             Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _RdLine()
*
*  Function description
*    Read text line terminated by newline.
*
*  Parameters
*    pSession - Pointer to SSL session.
*    pData    - Pointer to object that receives the data.
*    DataLen  - Octet length of the receiving object.
*
*  Return value
*    >= 0 - Success, number of octets received including newline.
*     < 0 - Failure.
*/
static int _RdLine(SSL_SESSION *pSession, U8 *pData, unsigned DataLen) {
  unsigned Len;
  int      Status;
  U8       Char;
  //
  Len = 0;
  for (;;) {
    Status = SSL_SESSION_Receive(pSession, &Char, 1);
    if (Status == 0) {
      return SSL_ERROR_EOF;
    } else if (Status < 0) {
      return Status;
    }
    pData[Len] = Char;
    if (Len+1 < DataLen) {
      ++Len;
    }
    if (Char == '\n') {
      pData[Len] = 0;
      return Len;
    }
  }
}

/*********************************************************************
*
*       _ApplyROT13()
*
*  Function description
*    Apply ROT13 transform.
*
*  Parameters
*    pData    - Pointer to object to transform.
*    DataLen  - Octet length of the object to transform.
*/
static void _ApplyROT13(U8 *pData, unsigned DataLen) {
  unsigned i;
  //
  for (i = 0; i < DataLen; ++i) {
    if ('a' <= pData[i] && pData[i] <= 'm') {
      pData[i] = pData[i] - 'a' + 'n';
    } else if ('n' <= pData[i] && pData[i] <= 'z') {
      pData[i] = pData[i] - 'n' + 'a';
    } else if ('A' <= pData[i] && pData[i] <= 'M') {
      pData[i] = pData[i] - 'A' + 'N';
    } else if ('N' <= pData[i] && pData[i] <= 'Z') {
      pData[i] = pData[i] - 'N' + 'A';
    }
  }
}

/*********************************************************************
*
*       _Serve()
*
*  Function description
*    Process a single ROT13 line.
*
*  Parameters
*    pSession - Pointer to SSL session.
*
*  Return value
*    >= 0 - Success, number of characters read, session remains open.
*    <  0 - Session closed.
*/
static int _Serve(SSL_SESSION *pSession) {
  char aData[256];
  int  Status;
  //
  Status = _RdLine(pSession, aData, sizeof(aData));
  if (Status >= 0) {
    SEGGER_SYS_IO_Printf("Recv: %s", aData);
    _ApplyROT13(aData, Status);
    Status = SSL_SESSION_Send(pSession, &aData[0], Status);
    if (Status >= 0) {
      SEGGER_SYS_IO_Printf("Sent: %s", aData);
    } else {
      SEGGER_SYS_IO_Printf("Error sending data: %s\n", SSL_ERROR_GetText(Status));
    }
  } else {
    if (Status != SSL_ERROR_EOF) {
      SEGGER_SYS_IO_Printf("Error receiving data: %s\n", SSL_ERROR_GetText(Status));
    }
  }
  //
  return Status;
}
```

```
/*********************************************************************
*
*             Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void);
void MainTask(void) {
  SSL_SESSION Session;
  int         BoundSocket;
  int         Socket;
  int         Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSL_Init();
  //
  SEGGER_SYS_IO_Printf("\n");
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n", SSL_GetCopyrightText());
  SEGGER_SYS_IO_Printf("emSSL ROT13 Server ");
  SEGGER_SYS_IO_Printf("compiled " __DATE__ " " __TIME__ "\n\n");
  //
  // Bind application's ROT13 port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(19000);
  if (BoundSocket < 0) {
    SEGGER_SYS_OS_Halt(100);
  }
  //
  for (;;) {
    //
    do {
      Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    } while (Socket < 0);
    //
    SSL_SESSION_Prepare(&Session, Socket, &_IP_Transport);
    Status = SSL_SESSION_Accept(&Session);
    //
    if (Status < 0) {
      SEGGER_SYS_IO_Printf("Can't negotiate a secure connection.\n\n");
      SEGGER_SYS_IP_Close(Socket);
    } else {
      do {
        Status = _Serve(&Session);
      } while (Status >= 0);
      SSL_SESSION_Disconnect(&Session);
      SEGGER_SYS_IP_CloseWait(Socket);
    }
  }
}

/*************************** End of file ****************************/
```

3.4 ROT13 client

The second application, `SSL_ROT13Client.c`, uses the secure ROT13 server to transform lines of text to their ROT13-encoded equivalent.

The process can be broken down into a sequence of steps:

- Initialize IP networking and SSL.
- Connect a plain socket to the ROT13 port on the remote server.
- Upgrade the socket to secure by negotiating a TLS session.
- Send data to be transformed to the server.
- Receive transformed data from the server.
- Close the connection and exit.

For a complete listing of this application, see *`SSL_ROT13Client.c` complete listing* on page 39.

3.4.1 Application entry

The main application task is responsible for setting up the environment ready to make outgoing SSL requests. This is simply boilerplate code that has no configuration:

```
void MainTask(void) {
  SSL_SESSION Session;
  int         Socket;
  //
  // Kick off networking and start SSL.
  //
  SEGGER_SYS_Init();  ❶
  SEGGER_SYS_IP_Init();
  SSL_Init();  ❷
  //
  SEGGER_SYS_IO_Printf("\n");  ❸
  SEGGER_SYS_IO_Printf("(c) 2017 SEGGER Microcontroller GmbH & Co. KG"
                       "    www.segger.com\n");
  SEGGER_SYS_IO_Printf("emSSL ROT13 Client ");
  SEGGER_SYS_IO_Printf("compiled " __DATE__ " " __TIME__ "\n\n");
```

❶ Initialize system components

The calls to `SEGGER_SYS_Init()` and `SEGGER_SYS_IP_Init()` use the SEGGER system abstraction layer to initialize services to the application and are identical to the server application.

❷ Initialize SSL component

The call to `SSL_Init()` initialized emSSL for use and is identical to the server application.

❸ Display identification

When running the client, this code just shows that the client is up and ready to make connections.

3.4.2 Making SSL connections

With the application initialized, the code progresses to interact with the server:

```
Socket = SEGGER_SYS_IP_Open(ROT13_SERVER, ROT13_PORT);  ❶
if (Socket < 0) {
  SEGGER_SYS_IO_Printf("Cannot open %s:%d!\n", ROT13_SERVER, ROT13_PORT);
  SEGGER_SYS_OS_Halt(100);
}
//
SSL_SESSION_Prepare(&Session, Socket, &_IP_Transport);  ❷
```

```
if (SSL_SESSION_Connect(&Session, ROT13_SERVER) < 0) {
  SEGGER_SYS_IO_Printf("Cannot negotiate a secure connection to %s:%d!\n",
                       ROT13_SERVER, ROT13_PORT);
  SEGGER_SYS_OS_Halt(100);
}
//
_RequestROT13(&Session, "SEGGER - The Embedded Experts\n");   ❸
_RequestROT13(&Session, "FRTTRE - Gur Rzorqqrq Rkcregf\n");
_RequestROT13(&Session, "SEGGER - It simply works!\n");
_RequestROT13(&Session, "FRTTRE - Vg fvzcyl jbexf!\n");
//
// Close the SSL connection.
//
SSL_SESSION_Disconnect(&Session);   ❹
SEGGER_SYS_IP_Close(Socket);
//
SSL_Exit();   ❺
SEGGER_SYS_IP_Exit();
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
```

❶ Open plain socket to server

The socket connection function is `SEGGER_SYS_IP_Open()` which is provided in `SEGGER_SYS.h`. This function will resolve a host's domain name and attempt to open a socket to the given port on the server. If everything goes without problems, the function result is a socket handle. If things go badly and the host is unreachable, or the port is refused, the function result is negative indicating an error.

By default the application opens a socket on the local host:

```
#define ROT13_SERVER  "127.0.0.1"
#define ROT13_PORT    19000
```

❷ Upgrade the socket to secure

To upgrade an open socket to secure, you use `SSL_SESSION_Prepare()` and `SSL_SESSION_Connect()`. The function `SSL_SESSION_Prepare()` will prepare a session and allow configuration of options before you negotiate an SSL connection using `SSL_SESSION_Connect()`. The function `SSL_SESSION_Connect()` returns a negative value if an SSL session cannot be established. An SSL connection may fail if the two peers cannot negotiate a common cipher suite or a common SSL version.

❸ Communicate with service

The application sends four separate requests to the ROT13 server for processing. The function `_RequestROT13()` which communicates with the server is presented below.

❹ Close the connection

Once you have finished with an SSL connection, or the peer closes the connection, you must release the resources associated with the connection using `SSL_SESSION_Disconnect()`. SSL disconnection terminates the connection between the peers but does not close the underlying socket. In order to close the socket that was established as the transport, you use `SYS_IP_Close`.

❺ Close SSL and IP

Closing down IP and SSL releases any resources that they hold.

3.4.3 Sending requests

The function `_RequestROT13()` send a request to the ROT13 server and accepts its reponse:

```
static void _RequestROT13(SSL_SESSION *pSession, const char *pData) {  ❶
  U8  aResponse[256];
  int Status;
  //
  // Send data to server.
  //
  Status = SSL_SESSION_SendStr(pSession, (const U8 *)pData);  ❷
  if (Status >= 0) {
    SEGGER_SYS_IO_Printf("Sent: %s", pData);
    Status = _RdLine(pSession, aResponse, sizeof(aResponse));  ❸
    if (Status >= 0) {
      SEGGER_SYS_IO_Printf("Recv: %s", aResponse);
    } else {
      SEGGER_SYS_IO_Printf("Error receiving data: %s\n",
                           SSL_ERROR_GetText(Status));
    }
  } else {
    SEGGER_SYS_IO_Printf("Error sending data: %s\n", SSL_ERROR_GetText(Status));
  }
}
```

❶ Accept parameters

The parameters are the connected SSL session and the zero-terminated string to transform.

❷ Send request

The zero-terminated string is sent to the server using `SSL_SESSION_SendStr()`. If it was not zero terminated, the function `SSL_SESSION_Send()` takes a "compound parameter" that is the data to send and its length.

❸ Read response

The response is read by _RdLine() which is identical to the code in the ROT13 server.

All remainng code is for processing errors and echoing data sent and received and is not further explained.

3.4.4 SSL_ROT13Client.c complete listing

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : SSL_ROT13Client.c
Purpose     : Simple client that uses a secure ROT13 service.

*/

/*********************************************************************
*
*       #include section
*
**********************************************************************
*/

#include "SSL.h"
#include "SEGGER_SYS.h"

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define ROT13_SERVER  "127.0.0.1"
#define ROT13_PORT    19000

/*********************************************************************
*
*             Static const data
*
**********************************************************************
*/

static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
};

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _RdLine()
*
*  Function description
*    Read text line terminated by newline.
*
*  Parameters
*    pSession - Pointer to SSL session.
*    pData    - Pointer to object that receives the data.
*    DataLen  - Octet length of the receiving object.
*
*  Return value
*    >= 0 - Success, number of octets received including newline.
*     < 0 - Failure.
*/
static int _RdLine(SSL_SESSION *pSession, U8 *pData, unsigned DataLen) {
  unsigned Len;
  int      Status;
  U8       Char;
  //
  Len = 0;
  for (;;) {
    Status = SSL_SESSION_Receive(pSession, &Char, 1);  ❶
    if (Status == 0) {  ❷
      return SSL_ERROR_EOF;
    } else if (Status < 0) {  ❸
      return Status;
    }
    pData[Len] = Char;  ❹
    if (Len+1 < DataLen) {
      ++Len;
    }
    if (Char == '\n') {  ❺
      pData[Len] = 0;
      return Len;
    }
  }
}

/*********************************************************************
*
*       _RequestROT13()
*
*  Function description
*    Apply ROT13 transform using ROT13 server.
*
*  Parameters
*    pSession - Pointer to SSL session.
*    pData    - Pointer to text to transform.
*/
static void _RequestROT13(SSL_SESSION *pSession, const char *pData) {
  U8  aResponse[256];
  int Status;
  //
  // Send data to server.
  //
  Status = SSL_SESSION_SendStr(pSession, (const U8 *)pData);
  if (Status >= 0) {
    SEGGER_SYS_IO_Printf("Sent: %s", pData);
    Status = _RdLine(pSession, aResponse, sizeof(aResponse));
    if (Status >= 0) {
      SEGGER_SYS_IO_Printf("Recv: %s", aResponse);
    } else {
      SEGGER_SYS_IO_Printf("Error receiving data: %s\n", SSL_ERROR_GetText(Status));
    }
  } else {
    SEGGER_SYS_IO_Printf("Error sending data: %s\n", SSL_ERROR_GetText(Status));
  }
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Ask ROT13 service to transform something.
*/
void MainTask(void);
void MainTask(void) {
  SSL_SESSION Session;
  int         Socket;
  //
  // Kick off networking and start SSL.
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSL_Init();
  //
  SEGGER_SYS_IO_Printf("\n");
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n", SSL_GetCopyrightText());
  SEGGER_SYS_IO_Printf("emSSL ROT13 Client ");
  SEGGER_SYS_IO_Printf("compiled " __DATE__ " " __TIME__ "\n\n");
  //
  // Open a plain socket to the server.
  //
  Socket = SEGGER_SYS_IP_Open(ROT13_SERVER, ROT13_PORT);
  if (Socket < 0) {
    SEGGER_SYS_IO_Printf("Cannot open %s:%d!\n", ROT13_SERVER, ROT13_PORT);
    SEGGER_SYS_OS_Halt(100);
  }
  //
  // Upgrade the connection to secure by negotiating a
  // session using SSL.
  //
  SSL_SESSION_Prepare(&Session, Socket, &_IP_Transport);
  if (SSL_SESSION_Connect(&Session, ROT13_SERVER) < 0) {
    SEGGER_SYS_IO_Printf("Cannot negotiate a secure connection to %s:%d!\n",
                         ROT13_SERVER, ROT13_PORT);
    SEGGER_SYS_OS_Halt(100);
  }
  //
  // We have established a secure connection, so send the server
  // some data.
  //
  _RequestROT13(&Session, "SEGGER - The Embedded Experts\n");
  _RequestROT13(&Session, "FRTTRE - Gur Rzorqqrq Rkcregf\n");
  _RequestROT13(&Session, "SEGGER - It simply works!\n");
  _RequestROT13(&Session, "FRTTRE - Vg fvzcyl jbexf!\n");
  //
  // Close the SSL connection.
  //
  SSL_SESSION_Disconnect(&Session);
  SEGGER_SYS_IP_Close(Socket);
  //
  // Finish up.
  //
  SSL_Exit();
  SEGGER_SYS_IP_Exit();
  SEGGER_SYS_OS_PauseBeforeHalt();
  SEGGER_SYS_OS_Halt(0);
}

/*************************** End of file ****************************/
```

3.5 Certificates

SSL uses a Public Key Infrastructure (PKI) to provide a chain of trust. The links in the chain are X.509 certificates which provide a trusted chain from the server's certificate to a root, trusted certificate.

It's beyond the scope of this document to describe how to acquire a certificate for a server hosted on the Internet, but we do describe how you integrate these certificates into emSSL such that your server can function securely on the Internet.

You will need to install a certificates when:

- You are using emSSL as a server, or
- You are using emSSL as a client but the server requires mutual authentication.

3.5.1 Types of certificate

There are three types of certificate that a Internet-facing server can provide:

- RSA certificate
- DSA certificate
- ECDSA certificate

You need to install certificates appropriate to the key agreement schemes that your server is configured to support.

RSA certificates are by far the most common type of public key certificate in use on servers today: certificate authorities issue them and they are universally accepted. There are arguments for and against both DSA and ECDSA certificates, and you should take some time to understand the advantages, disadvantages, and potential hazards with these.

Note

We recommend that you use RSA certificates when configuring emSSL to avoid any potential incompatibilities with SSL clients. It is beyond the scope of this document to describe the merits of RSA certificates as opposed to DSA and ECDSA certificates.

3.5.2 Self-signed certificates

In the following sections, we will describe how to configure emSSL with certificates by using OpenSSL to create *self-signed certificates*. A self-signed certificate has no chain of trust to a well-known certificate authority, it stands by itself, which is a great advantage when testing out emSSL: you don't have to wait for a CA to issue you a certificate for testing.

Using a self-signed certificate for a web server, on an intranet or on the Internet, will cause warnings from all good web browsers. You'll typically be asked whether you want to trust the connection by accepting the certificate. For servers that you don't control, you would decline the certificate, but for your own test servers, it's just fine to accept the certificate that you created and signed.

3.6 Creating certificates using OpenSSL

OpenSSL has the ability to generate X.509 certificates in multiple formats. This section describes how to use OpenSSL to create self-signed certificates that you can install into emSSL when running emSSL as a TLS server.

OpenSSL comes preinstalled on Mac OS X and with many Linux distributions. You may wish to use Windows binaries, in which case you will find appropriate information here:

https://www.openssl.org/related/binaries.html

We describe how to use OpenSSL using Mac OS X, but the steps are the same on Linux and Windows.

3.6.1 Creating RSA certificates

The process for creating an RSA certificate has two steps:

- Create a private and public key pair. It is imperative that the private key remains private and that you secure it against theft.
- Publish the public key, together with identification information, as an X.509 RSA certificate. The public key is, by definition, public and does not require protection.

Generate the private key

First, generate a private RSA key file. The private key file contains the generated RSA private key for signing and the public key for signature verification and in this example we request a modulus length of 2048 bits:

```
MacBook:~ paul$ openssl genrsa -out rsakey.pem 2048
Generating RSA private key, 2048 bit long modulus
...+++
..................................+++
e is 65537 (0x10001)
MacBook:~ paul$ _
```

The file `rsakey.pem` contains an unencrypted private key that is used for signing combined with an unencrypted public key that is used for signature verification. Longer key sizes (with larger moduli) offer increased security. The recommendation at the time of publication is to use an RSA modulus of no less than 1024 bits, with 2048 bits being preferred.

Generate the certificate

We combine the public key from the key file, together with identity information, into a self-signed certificate for emSSL:

```
MacBook:~ paul$ openssl req -new -x509 -key rsakey.pem -outform DER -out
 rsacert.der
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Nordrhein-Westfalen
Locality Name (eg, city) []:Hilden
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SEGGER Microcontroller
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:emssl.segger.com
Email Address []:
MacBook:~ paul$ _
```

The file `rsacert.der` is a self-signed DER-encoded certificate that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

Prepare the private keys

Now we need to prepare the corresponding private key that the server will use to sign data during key exchange. The RSA keys were written in PEM format, but must be presented in DER format, so we need to convert them:

```
MacBook:~ paul$ openssl rsa -in rsakey.pem -outform DER -out rsakey.der
writing RSA key
MacBook:~ paul$ _
```

The file `rsakey.der` is a DER-encoded private key that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

3.6.2 Creating DSA certificates

The process of generating a DSA certificate is similar to generating an RSA certificate except that DSA has the ability to share domain parameters between users. We won’t discuss this particular feature here, we will simply explain what you need to do in order to create a DSA certificate.

Generate the private key

First, generate a private DSA key file which is a two-stage process:

- Generate a set of DSA parameters, called a DSA domain
- Using the DSA domain, generate a private DSA key

First, ask OpenSSL to generate a set of DSA parameters using a strong 2048-bit prime:

```
MacBook:~ paul$ openssl dsaparam -out dsaparam.pem 2048
Generating DSA parameters, 2048 bit long prime
This could take some time
.......................+++++++++++++++++++++++++++++++++++++++++++++++++++*
.+.........+..+.........+.....+...+......+...+++++++++++++++++++++++++++++++
+++++++++++++++++++++*
MacBook:~ paul$ _
```

This generates a set of DSA parameters that can be shared between users. In this case we will not be sharing the parameters, but they are still required to generate the private key. As with RSA, security scales with the length of the key.

Note

The emSSL DSA implementation follows the NIST standard which means that key lengths are limited: 1024, 2048, and 3072 bits. Sizes other than this will result in key files and certificates that are unusable, signaled by error statuses returned by the emSSL API.

Next, generate a DSA private key file. The private key file contains the private key for signing:

```
MacBook:~ paul$ openssl gendsa -out dsakey.pem dsaparam.pem
Generating DSA key, 2048 bits
MacBook:~ paul$ _
```

The file dsakey.pem now contains an unencrypted private key that can be used for signing.

Generate the certificate

We wrap the private key from the key file, together with identity information, to generate a self-signed certificate for emSSL:

```
MacBook:~ paul$ openssl req -new -x509-nodes -key dsakey.pem -outform DER
-out dsacert.der
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Nordrhein-Westfalen
Locality Name (eg, city) []:Hilden
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SEGGER Microcontroller
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:emssl.segger.com
Email Address []:
MacBook:~ paul$ _
```

The file `dsacert.der` is a self-signed DER-encoded certificate that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

Prepare the private keys

Now we need to prepare the DSA private key that the server will use to sign data during key exchange. The DSA keys were written in PEM format, but need to be presented in DER format, so we must convert them:

```
MacBook:~ paul$ openssl dsa -in dsakey.pem -outform DER -out dsakey.der
writing RSA key
MacBook:~ paul$
```

The file `dsakey.der` is a DER-encoded private key that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

3.6.3 Creating ECDSA certificates

The process of generating an ECDSA certificate is similar to generating an RSA certificate; no, this is not a mistake, the OpenSSL steps look more like RSA than DSA. The technical difference between ECDSA and DSA is the mathematical basis that underpins the signature and verification algorithms and the choice of key parameters.

Generate the private key

First, generate a private ECDSA key file using a strong 224-bit curve:

```
MacBook:~ paul$ openssl ecparam -out ecparam.pem -name secp224r1 -genkey
MacBook:~ paul$ _
```

This generates a set of ECDSA parameters with key.

The curves that are common to emSSL and OpenSSL are:

- NIST P-224 which is known as secp224r1.
- NIST P-384 which is known as secp384r1.
- NIST P-521 which is known as secp521r1.

Both emSSL and OpenSSL support more curves than this, but the NIST curves have the advantage that they are standardized (however only three curves are common to both implementations).

Generate the certificate

We wrap the private key from the key file, together with identity information, to generate a self-signed certificate for emSSL:

```
MacBook:~ paul$ openssl req -new -x509 -nodes -key eckey.pem -outform DER -out
 eccert.der
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Nordrhein-Westfalen
Locality Name (eg, city) []:Hilden
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SEGGER Microcontroller
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:emssl.segger.com
Email Address []:
MacBook:~ paul$ _
```

The file `eccert.der` is a self-signed DER-encoded certificate that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

Prepare the private keys

Now we need to prepare the ECDSA private key that the server will use to sign data during key exchange. The ECDSA keys were written in PEM format, but need to be presented in DER format, so we must convert them:

```
MacBook:~ paul$ openssl ec -in eckey.pem -outform DER -out eckey.der
read EC key
writing EC key
MacBook:~ paul$
```

The file `eckey.der` is a DER-encoded private key that needs to be installed into the emSSL server, covered in *Installing certificates and keys* on page 47.

3.7 Installing certificates and keys

Once you have generated your certificates and keys in DER format, you must install them into emSSL. Typically you would store these such that they can be replaced just before they expire, for example on a file system managed by emFile. For simplicity, in this example, we will embed the RSA certificate and private key into the application as read-only arrays.

3.7.1 Installing a single RSA certificate and key

The SEGGER utility `Bin2C` supplied with emSSL will take a file and convert it to a C array:

```
C:> bin2c rsacert.der rsacert
C:> bin2c rsakey.der rsakey
C:> _
```

After running this command, the file `rsacert.h` contains a C array declaration and the file `rsacert.c` contains the corresponding definition:

```
/*
  C-file generated by Bin2C
  Compiled:    Aug  8 2014 at 14:50:47

  Copyright (C) 2013
  Segger Microcontroller GmbH & Co. KG
  www.segger.com

  Solutions for real time microcontroller applications
*/

static const unsigned char _rsacert[1352UL + 1] = {
  0x30, 0x82, 0x05, 0x44, 0x30, 0x82, 0x05, 0x02, 0xA0, 0x03, 0x02, 0x01, 0x02,
  ...
```

To integrate certificate and private key into emSSL, you provide an implementation for the certificate API. The certificate API has three functions that you must provide, one for certificate verification, one for retrieving a certificate, and one for retrieving the certificate’s private key:

```
static const SSL_CERTIFCATE_API _CertificateAPI = {
  NULL, /* Use default certificate verification */
  _GetCertificate,
  _GetPrivateKey,
};
```

In this example, we have a single RSA self-signed certificate and, therefore, we can only support key exchanges using RSA cipher suites. The implementation of the certificate function is straightforward:

```
static int _GetCertificate(SSL_SESSION * pSession,
                           unsigned       Index,
                           const U8    ** ppData,
                           unsigned     * pLen) {
  //
  // We only support a single self-signed certificate and
  // corresponding private key.
  //
  if (Index == 0) {
    *ppData = rsacert_file;
    *pLen   = RSACERT_SIZE;
    return 0;
  } else {
    *ppData = 0;
    *pLen   = 0;
    return -1;
  }
}
```

And so is the corresponding private key:

```
static int _GetPrivateKey(SSL_SESSION * pSession,
                          const U8   ** ppData,
                          unsigned    * pDataLen) {
  //
  // We only support a single self-signed certificate and
  // corresponding private key.
  //
  *ppData    = rsakey_file;
  *pDataLen = RSAKEY_SIZE;
  return 0;
}
```

The certificate API needs to be set for each session individually using `SSL_SESSION_SetCertificateAPI`:

```
SSL_SESSION_SetCertificateAPI(&Session, &_CertificateAPI);
```

3.7.2 Installing multiple certificate types

Because emSSL supports different key agreement protocols, it may well be necessary for you to install more than one type of certificate. In this case, the certificate that emSSL will serve will depend upon the cipher suite that is agreed between the client and server.

The two most common key agreement protocols in use today use RSA and Elliptic Curve Diffie-Hellman (to provide forward secrecy). For elliptic curve suites, you must provide an elliptic curve certificate and use a corresponding private key.

When emSSL asks for a certificate, you can examine the cipher suite that has been agreed, and propose a certificate to use:

```
static int _GetCertificate(SSL_SESSION * pSession,
                           unsigned      Index,
                           const U8   ** ppData,
                           unsigned    * pLen) {
  //
  // We support a self-signed certificate and corresponding
  // private key in two forms.
  //
  if (Index == 0) {
    if (SSL_SUITE_QueryRequiresECC(SSL_SESSION_GetSuite(pSession))) {
      *ppData = eccert_file;
      *pLen   = ECCERT_SIZE;
    } else {
      *ppData = rsacert_file;
      *pLen   = RSACERT_SIZE;
    }
    return 0;
  } else {
    *ppData = 0;
    *pLen   = 0;
    return -1;
  }
}
```

The same is true for the corresponding private key:

```
static int _GetPrivateKey(SSL_SESSION * pSession,
                          const U8   ** ppData,
                          unsigned    * pDataLen) {
```

```
  //
  // We support a self-signed certificate and corresponding
  // private key in two forms.
  //
  if (SSL_SUITE_QueryRequiresECC(SSL_SESSION_GetSuite(pSession))) {
    *ppData    = eckey_file;
    *pDataLen = ECKEY_SIZE;
  } else {
    *ppData    = rsakey_file;
    *pDataLen = RSAKEY_SIZE;
  }
  //
  return 0;
}
```

3.7.3 Installing root certificates

In order to authenticate the host that you are connecting to, the SSL server provides a certificate chain that has a trusted root. It's common for web browsers to ship with a set of *trusted roots* which are automatically trusted when encountered.

By default emSSL has an empty trust store. To install trusted root certificates you must acquire these from the certificate authorities that you trust and use the `PrintCert` utility to convert the DER or PEM certificate to something that can be added to emSSL.

The file `Sample/Config/SSL/SSL_X_TrustedCerts.c` contains some preconverted root certificates from GeoTrust, GlobalSign, and VeriSign that are added to emSSL when running sample applications.

Note

We highly recommended that you source and convert your own selection of root certificates.

Example

The following shows how to convert a root certificate to a form that can be added to emSSL:

```
MacBook:~ paul$ PrintCert GeoTrust_Primary_CA.pem -p \
> SSL_CERTIFICATE_GeoTrust_Primary_CA >SSL_X_TrustedCerts.c

(c) 2015-2016 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
emSSL PrintCert V2.30 compiled Aug  2 2016 22:00:12

Subject: GeoTrust Primary Certification Authority
Issuer:  GeoTrust Primary Certification Authority

MacBook:~ paul$ _
```

3.7.4 Certificate conversion utility reference

The certificate conversion utility converts a PEM or DER certificate to a form usable by emSSL.

Usage

`PrintCert.exe` [<Options>] <file>

emSSL PrintCert accepts the following command line options:

Option	Description
-x	Declare object with external storage.

Option	Description
`-p` *string*	Set the object name prefix to *string*. Default is empty.

Chapter 4

API reference

This chapter explains the API functions of emSSL which are needed for secure communication. The emSSL API is kept as simple as possible to provide a straightforward way to integrate emSSL into a product.

4.1 Preprocessor symbols

4.1.1 Version number

Description

Symbol expands to a number that identifies the specific emSSL release.

Definition

```
#define SSL_VERSION    25600
```

Symbols

Definition	Description
SSL_VERSION	Format is “Mmmrr” so, for example, 25401 corresponds to version 2.54a.

4.1.2 Cipher suite IDs

Description

Official IANA names for cipher suites, but using emSSL’s “SSL” prefix rather than “TLS”.

Definition

```
#define SSL_SUITE_ID_NULL_WITH_NULL_NULL                          0x0000
#define SSL_SUITE_ID_RSA_WITH_NULL_MD5                            0x0001
#define SSL_SUITE_ID_RSA_WITH_NULL_SHA                            0x0002
#define SSL_SUITE_ID_RSA_EXPORT_WITH_RC4_40_MD5                   0x0003
#define SSL_SUITE_ID_RSA_WITH_RC4_128_MD5                         0x0004
#define SSL_SUITE_ID_RSA_WITH_RC4_128_SHA                         0x0005
#define SSL_SUITE_ID_RSA_EXPORT_WITH_RC2_CBC_40_MD5               0x0006
#define SSL_SUITE_ID_RSA_WITH_IDEA_CBC_SHA                        0x0007
#define SSL_SUITE_ID_RSA_EXPORT_WITH_DES40_CBC_SHA                0x0008
#define SSL_SUITE_ID_RSA_WITH_DES_CBC_SHA                         0x0009
#define SSL_SUITE_ID_RSA_WITH_3DES_EDE_CBC_SHA                    0x000A
#define SSL_SUITE_ID_DH_DSS_EXPORT_WITH_DES40_CBC_SHA             0x000B
#define SSL_SUITE_ID_DH_DSS_WITH_DES_CBC_SHA                      0x000C
#define SSL_SUITE_ID_DH_DSS_WITH_3DES_EDE_CBC_SHA                 0x000D
#define SSL_SUITE_ID_DH_RSA_EXPORT_WITH_DES40_CBC_SHA             0x000E
#define SSL_SUITE_ID_DH_RSA_WITH_DES_CBC_SHA                      0x000F
#define SSL_SUITE_ID_DH_RSA_WITH_3DES_EDE_CBC_SHA                 0x0010
#define SSL_SUITE_ID_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA            0x0011
#define SSL_SUITE_ID_DHE_DSS_WITH_DES_CBC_SHA                     0x0012
#define SSL_SUITE_ID_DHE_DSS_WITH_3DES_EDE_CBC_SHA                0x0013
#define SSL_SUITE_ID_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA            0x0014
#define SSL_SUITE_ID_DHE_RSA_WITH_DES_CBC_SHA                     0x0015
#define SSL_SUITE_ID_DHE_RSA_WITH_3DES_EDE_CBC_SHA                0x0016
#define SSL_SUITE_ID_DH_anon_EXPORT_WITH_RC4_40_MD5               0x0017
#define SSL_SUITE_ID_DH_anon_WITH_RC4_128_MD5                     0x0018
#define SSL_SUITE_ID_DH_anon_EXPORT_WITH_DES40_CBC_SHA            0x0019
#define SSL_SUITE_ID_DH_anon_WITH_DES_CBC_SHA                     0x001A
#define SSL_SUITE_ID_DH_anon_WITH_3DES_EDE_CBC_SHA                0x001B
#define SSL_SUITE_ID_KRB5_WITH_DES_CBC_SHA                        0x001E
#define SSL_SUITE_ID_KRB5_WITH_3DES_EDE_CBC_SHA                   0x001F
#define SSL_SUITE_ID_KRB5_WITH_RC4_128_SHA                        0x0020
#define SSL_SUITE_ID_KRB5_WITH_IDEA_CBC_SHA                       0x0021
#define SSL_SUITE_ID_KRB5_WITH_DES_CBC_MD5                        0x0022
#define SSL_SUITE_ID_KRB5_WITH_3DES_EDE_CBC_MD5                   0x0023
#define SSL_SUITE_ID_KRB5_WITH_RC4_128_MD5                        0x0024
#define SSL_SUITE_ID_KRB5_WITH_IDEA_CBC_MD5                       0x0025
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_DES_CBC_40_SHA              0x0026
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_RC2_CBC_40_SHA              0x0027
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_RC4_40_SHA                  0x0028
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_DES_CBC_40_MD5              0x0029
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_RC2_CBC_40_MD5              0x002A
#define SSL_SUITE_ID_KRB5_EXPORT_WITH_RC4_40_MD5                  0x002B
#define SSL_SUITE_ID_PSK_WITH_NULL_SHA                            0x002C
#define SSL_SUITE_ID_DHE_PSK_WITH_NULL_SHA                        0x002D
#define SSL_SUITE_ID_RSA_PSK_WITH_NULL_SHA                        0x002E
#define SSL_SUITE_ID_RSA_WITH_AES_128_CBC_SHA                     0x002F
#define SSL_SUITE_ID_DH_DSS_WITH_AES_128_CBC_SHA                  0x0030
#define SSL_SUITE_ID_DH_RSA_WITH_AES_128_CBC_SHA                  0x0031
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_128_CBC_SHA                 0x0032
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_128_CBC_SHA                 0x0033
#define SSL_SUITE_ID_DH_anon_WITH_AES_128_CBC_SHA                 0x0034
#define SSL_SUITE_ID_RSA_WITH_AES_256_CBC_SHA                     0x0035
#define SSL_SUITE_ID_DH_DSS_WITH_AES_256_CBC_SHA                  0x0036
#define SSL_SUITE_ID_DH_RSA_WITH_AES_256_CBC_SHA                  0x0037
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_256_CBC_SHA                 0x0038
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_256_CBC_SHA                 0x0039
#define SSL_SUITE_ID_DH_anon_WITH_AES_256_CBC_SHA                 0x003A
#define SSL_SUITE_ID_RSA_WITH_NULL_SHA256                         0x003B
#define SSL_SUITE_ID_RSA_WITH_AES_128_CBC_SHA256                  0x003C
```

```
#define SSL_SUITE_ID_RSA_WITH_AES_256_CBC_SHA256              0x003D
#define SSL_SUITE_ID_DH_DSS_WITH_AES_128_CBC_SHA256           0x003E
#define SSL_SUITE_ID_DH_RSA_WITH_AES_128_CBC_SHA256           0x003F
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_128_CBC_SHA256          0x0040
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_128_CBC_SHA            0x0041
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_128_CBC_SHA         0x0042
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_128_CBC_SHA         0x0043
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA        0x0044
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA        0x0045
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_128_CBC_SHA        0x0046
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_128_CBC_SHA256          0x0067
#define SSL_SUITE_ID_DH_DSS_WITH_AES_256_CBC_SHA256           0x0068
#define SSL_SUITE_ID_DH_RSA_WITH_AES_256_CBC_SHA256           0x0069
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_256_CBC_SHA256          0x006A
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_256_CBC_SHA256          0x006B
#define SSL_SUITE_ID_DH_anon_WITH_AES_128_CBC_SHA256          0x006C
#define SSL_SUITE_ID_DH_anon_WITH_AES_256_CBC_SHA256          0x006D
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_256_CBC_SHA            0x0084
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_256_CBC_SHA         0x0085
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_256_CBC_SHA         0x0086
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA        0x0087
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA        0x0088
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_256_CBC_SHA        0x0089
#define SSL_SUITE_ID_PSK_WITH_RC4_128_SHA                     0x008A
#define SSL_SUITE_ID_PSK_WITH_3DES_EDE_CBC_SHA                0x008B
#define SSL_SUITE_ID_PSK_WITH_AES_128_CBC_SHA                 0x008C
#define SSL_SUITE_ID_PSK_WITH_AES_256_CBC_SHA                 0x008D
#define SSL_SUITE_ID_DHE_PSK_WITH_RC4_128_SHA                 0x008E
#define SSL_SUITE_ID_DHE_PSK_WITH_3DES_EDE_CBC_SHA            0x008F
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_128_CBC_SHA             0x0090
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_256_CBC_SHA             0x0091
#define SSL_SUITE_ID_RSA_PSK_WITH_RC4_128_SHA                 0x0092
#define SSL_SUITE_ID_RSA_PSK_WITH_3DES_EDE_CBC_SHA            0x0093
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_128_CBC_SHA             0x0094
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_256_CBC_SHA             0x0095
#define SSL_SUITE_ID_RSA_WITH_SEED_CBC_SHA                    0x0096
#define SSL_SUITE_ID_DH_DSS_WITH_SEED_CBC_SHA                 0x0097
#define SSL_SUITE_ID_DH_RSA_WITH_SEED_CBC_SHA                 0x0098
#define SSL_SUITE_ID_DHE_DSS_WITH_SEED_CBC_SHA                0x0099
#define SSL_SUITE_ID_DHE_RSA_WITH_SEED_CBC_SHA                0x009A
#define SSL_SUITE_ID_DH_anon_WITH_SEED_CBC_SHA                0x009B
#define SSL_SUITE_ID_RSA_WITH_AES_128_GCM_SHA256              0x009C
#define SSL_SUITE_ID_RSA_WITH_AES_256_GCM_SHA384              0x009D
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_128_GCM_SHA256          0x009E
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_256_GCM_SHA384          0x009F
#define SSL_SUITE_ID_DH_RSA_WITH_AES_128_GCM_SHA256           0x00A0
#define SSL_SUITE_ID_DH_RSA_WITH_AES_256_GCM_SHA384           0x00A1
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_128_GCM_SHA256          0x00A2
#define SSL_SUITE_ID_DHE_DSS_WITH_AES_256_GCM_SHA384          0x00A3
#define SSL_SUITE_ID_DH_DSS_WITH_AES_128_GCM_SHA256           0x00A4
#define SSL_SUITE_ID_DH_DSS_WITH_AES_256_GCM_SHA384           0x00A5
#define SSL_SUITE_ID_DH_anon_WITH_AES_128_GCM_SHA256          0x00A6
#define SSL_SUITE_ID_DH_anon_WITH_AES_256_GCM_SHA384          0x00A7
#define SSL_SUITE_ID_PSK_WITH_AES_128_GCM_SHA256              0x00A8
#define SSL_SUITE_ID_PSK_WITH_AES_256_GCM_SHA384              0x00A9
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_128_GCM_SHA256          0x00AA
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_256_GCM_SHA384          0x00AB
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_128_GCM_SHA256          0x00AC
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_256_GCM_SHA384          0x00AD
#define SSL_SUITE_ID_PSK_WITH_AES_128_CBC_SHA256              0x00AE
#define SSL_SUITE_ID_PSK_WITH_AES_256_CBC_SHA384              0x00AF
#define SSL_SUITE_ID_PSK_WITH_NULL_SHA256                     0x00B0
#define SSL_SUITE_ID_PSK_WITH_NULL_SHA384                     0x00B1
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_128_CBC_SHA256          0x00B2
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_256_CBC_SHA384          0x00B3
#define SSL_SUITE_ID_DHE_PSK_WITH_NULL_SHA256                 0x00B4
#define SSL_SUITE_ID_DHE_PSK_WITH_NULL_SHA384                 0x00B5
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_128_CBC_SHA256          0x00B6
```

```
#define SSL_SUITE_ID_RSA_PSK_WITH_AES_256_CBC_SHA384               0x00B7
#define SSL_SUITE_ID_RSA_PSK_WITH_NULL_SHA256                      0x00B8
#define SSL_SUITE_ID_RSA_PSK_WITH_NULL_SHA384                      0x00B9
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_128_CBC_SHA256              0x00BA
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256           0x00BB
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256           0x00BC
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256          0x00BD
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256          0x00BE
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_128_CBC_SHA256          0x00BF
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_256_CBC_SHA256              0x00C0
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256           0x00C1
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256           0x00C2
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256          0x00C3
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256          0x00C4
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_256_CBC_SHA256          0x00C5
#define SSL_SUITE_ID_EMPTY_RENEGOTIATION_INFO_SCSV                 0x00FF
#define SSL_SUITE_ID_FALLBACK_SCSV                                 0x5600
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_NULL_SHA                      0xC001
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_RC4_128_SHA                   0xC002
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA              0xC003
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_128_CBC_SHA               0xC004
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_256_CBC_SHA               0xC005
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_NULL_SHA                     0xC006
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_RC4_128_SHA                  0xC007
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA             0xC008
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_128_CBC_SHA              0xC009
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_256_CBC_SHA              0xC00A
#define SSL_SUITE_ID_ECDH_RSA_WITH_NULL_SHA                        0xC00B
#define SSL_SUITE_ID_ECDH_RSA_WITH_RC4_128_SHA                     0xC00C
#define SSL_SUITE_ID_ECDH_RSA_WITH_3DES_EDE_CBC_SHA                0xC00D
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_128_CBC_SHA                 0xC00E
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_256_CBC_SHA                 0xC00F
#define SSL_SUITE_ID_ECDHE_RSA_WITH_NULL_SHA                       0xC010
#define SSL_SUITE_ID_ECDHE_RSA_WITH_RC4_128_SHA                    0xC011
#define SSL_SUITE_ID_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA               0xC012
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_128_CBC_SHA                0xC013
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_256_CBC_SHA                0xC014
#define SSL_SUITE_ID_ECDH_anon_WITH_NULL_SHA                       0xC015
#define SSL_SUITE_ID_ECDH_anon_WITH_RC4_128_SHA                    0xC016
#define SSL_SUITE_ID_ECDH_anon_WITH_3DES_EDE_CBC_SHA               0xC017
#define SSL_SUITE_ID_ECDH_anon_WITH_AES_128_CBC_SHA                0xC018
#define SSL_SUITE_ID_ECDH_anon_WITH_AES_256_CBC_SHA                0xC019
#define SSL_SUITE_ID_SRP_SHA_WITH_3DES_EDE_CBC_SHA                 0xC01A
#define SSL_SUITE_ID_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA             0xC01B
#define SSL_SUITE_ID_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA             0xC01C
#define SSL_SUITE_ID_SRP_SHA_WITH_AES_128_CBC_SHA                  0xC01D
#define SSL_SUITE_ID_SRP_SHA_RSA_WITH_AES_128_CBC_SHA              0xC01E
#define SSL_SUITE_ID_SRP_SHA_DSS_WITH_AES_128_CBC_SHA              0xC01F
#define SSL_SUITE_ID_SRP_SHA_WITH_AES_256_CBC_SHA                  0xC020
#define SSL_SUITE_ID_SRP_SHA_RSA_WITH_AES_256_CBC_SHA              0xC021
#define SSL_SUITE_ID_SRP_SHA_DSS_WITH_AES_256_CBC_SHA              0xC022
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256           0xC023
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384           0xC024
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_128_CBC_SHA256            0xC025
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_256_CBC_SHA384            0xC026
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_128_CBC_SHA256             0xC027
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_256_CBC_SHA384             0xC028
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_128_CBC_SHA256              0xC029
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_256_CBC_SHA384              0xC02A
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256           0xC02B
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384           0xC02C
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_128_GCM_SHA256            0xC02D
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_AES_256_GCM_SHA384            0xC02E
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_128_GCM_SHA256             0xC02F
#define SSL_SUITE_ID_ECDHE_RSA_WITH_AES_256_GCM_SHA384             0xC030
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_128_GCM_SHA256              0xC031
#define SSL_SUITE_ID_ECDH_RSA_WITH_AES_256_GCM_SHA384              0xC032
#define SSL_SUITE_ID_ECDHE_PSK_WITH_RC4_128_SHA                    0xC033
```

```
#define SSL_SUITE_ID_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA           0xC034
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_128_CBC_SHA            0xC035
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_256_CBC_SHA            0xC036
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_128_CBC_SHA256         0xC037
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_256_CBC_SHA384         0xC038
#define SSL_SUITE_ID_ECDHE_PSK_WITH_NULL_SHA                   0xC039
#define SSL_SUITE_ID_ECDHE_PSK_WITH_NULL_SHA256                0xC03A
#define SSL_SUITE_ID_ECDHE_PSK_WITH_NULL_SHA384                0xC03B
#define SSL_SUITE_ID_RSA_WITH_ARIA_128_CBC_SHA256              0xC03C
#define SSL_SUITE_ID_RSA_WITH_ARIA_256_CBC_SHA384              0xC03D
#define SSL_SUITE_ID_DH_DSS_WITH_ARIA_128_CBC_SHA256           0xC03E
#define SSL_SUITE_ID_DH_DSS_WITH_ARIA_256_CBC_SHA384           0xC03F
#define SSL_SUITE_ID_DH_RSA_WITH_ARIA_128_CBC_SHA256           0xC040
#define SSL_SUITE_ID_DH_RSA_WITH_ARIA_256_CBC_SHA384           0xC041
#define SSL_SUITE_ID_DHE_DSS_WITH_ARIA_128_CBC_SHA256          0xC042
#define SSL_SUITE_ID_DHE_DSS_WITH_ARIA_256_CBC_SHA384          0xC043
#define SSL_SUITE_ID_DHE_RSA_WITH_ARIA_128_CBC_SHA256          0xC044
#define SSL_SUITE_ID_DHE_RSA_WITH_ARIA_256_CBC_SHA384          0xC045
#define SSL_SUITE_ID_DH_anon_WITH_ARIA_128_CBC_SHA256          0xC046
#define SSL_SUITE_ID_DH_anon_WITH_ARIA_256_CBC_SHA384          0xC047
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256      0xC048
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384      0xC049
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256       0xC04A
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384       0xC04B
#define SSL_SUITE_ID_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256        0xC04C
#define SSL_SUITE_ID_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384        0xC04D
#define SSL_SUITE_ID_ECDH_RSA_WITH_ARIA_128_CBC_SHA256         0xC04E
#define SSL_SUITE_ID_ECDH_RSA_WITH_ARIA_256_CBC_SHA384         0xC04F
#define SSL_SUITE_ID_RSA_WITH_ARIA_128_GCM_SHA256              0xC050
#define SSL_SUITE_ID_RSA_WITH_ARIA_256_GCM_SHA384              0xC051
#define SSL_SUITE_ID_DHE_RSA_WITH_ARIA_128_GCM_SHA256          0xC052
#define SSL_SUITE_ID_DHE_RSA_WITH_ARIA_256_GCM_SHA384          0xC053
#define SSL_SUITE_ID_DH_RSA_WITH_ARIA_128_GCM_SHA256           0xC054
#define SSL_SUITE_ID_DH_RSA_WITH_ARIA_256_GCM_SHA384           0xC055
#define SSL_SUITE_ID_DHE_DSS_WITH_ARIA_128_GCM_SHA256          0xC056
#define SSL_SUITE_ID_DHE_DSS_WITH_ARIA_256_GCM_SHA384          0xC057
#define SSL_SUITE_ID_DH_DSS_WITH_ARIA_128_GCM_SHA256           0xC058
#define SSL_SUITE_ID_DH_DSS_WITH_ARIA_256_GCM_SHA384           0xC059
#define SSL_SUITE_ID_DH_anon_WITH_ARIA_128_GCM_SHA256          0xC05A
#define SSL_SUITE_ID_DH_anon_WITH_ARIA_256_GCM_SHA384          0xC05B
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256      0xC05C
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384      0xC05D
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256       0xC05E
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384       0xC05F
#define SSL_SUITE_ID_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256        0xC060
#define SSL_SUITE_ID_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384        0xC061
#define SSL_SUITE_ID_ECDH_RSA_WITH_ARIA_128_GCM_SHA256         0xC062
#define SSL_SUITE_ID_ECDH_RSA_WITH_ARIA_256_GCM_SHA384         0xC063
#define SSL_SUITE_ID_PSK_WITH_ARIA_128_CBC_SHA256              0xC064
#define SSL_SUITE_ID_PSK_WITH_ARIA_256_CBC_SHA384              0xC065
#define SSL_SUITE_ID_DHE_PSK_WITH_ARIA_128_CBC_SHA256          0xC066
#define SSL_SUITE_ID_DHE_PSK_WITH_ARIA_256_CBC_SHA384          0xC067
#define SSL_SUITE_ID_RSA_PSK_WITH_ARIA_128_CBC_SHA256          0xC068
#define SSL_SUITE_ID_RSA_PSK_WITH_ARIA_256_CBC_SHA384          0xC069
#define SSL_SUITE_ID_PSK_WITH_ARIA_128_GCM_SHA256              0xC06A
#define SSL_SUITE_ID_PSK_WITH_ARIA_256_GCM_SHA384              0xC06B
#define SSL_SUITE_ID_DHE_PSK_WITH_ARIA_128_GCM_SHA256          0xC06C
#define SSL_SUITE_ID_DHE_PSK_WITH_ARIA_256_GCM_SHA384          0xC06D
#define SSL_SUITE_ID_RSA_PSK_WITH_ARIA_128_GCM_SHA256          0xC06E
#define SSL_SUITE_ID_RSA_PSK_WITH_ARIA_256_GCM_SHA384          0xC06F
#define SSL_SUITE_ID_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256        0xC070
#define SSL_SUITE_ID_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384        0xC071
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256  0xC072
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384  0xC073
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256   0xC074
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384   0xC075
#define SSL_SUITE_ID_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256    0xC076
#define SSL_SUITE_ID_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384    0xC077
```

```
#define SSL_SUITE_ID_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256        0xC078
#define SSL_SUITE_ID_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384        0xC079
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_128_GCM_SHA256             0xC07A
#define SSL_SUITE_ID_RSA_WITH_CAMELLIA_256_GCM_SHA384             0xC07B
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256         0xC07C
#define SSL_SUITE_ID_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384         0xC07D
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256          0xC07E
#define SSL_SUITE_ID_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384          0xC07F
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256         0xC080
#define SSL_SUITE_ID_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384         0xC081
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256          0xC082
#define SSL_SUITE_ID_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384          0xC083
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_128_GCM_SHA256         0xC084
#define SSL_SUITE_ID_DH_anon_WITH_CAMELLIA_256_GCM_SHA384         0xC085
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256     0xC086
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384     0xC087
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256      0xC088
#define SSL_SUITE_ID_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384      0xC089
#define SSL_SUITE_ID_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256       0xC08A
#define SSL_SUITE_ID_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384       0xC08B
#define SSL_SUITE_ID_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256        0xC08C
#define SSL_SUITE_ID_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384        0xC08D
#define SSL_SUITE_ID_PSK_WITH_CAMELLIA_128_GCM_SHA256             0xC08E
#define SSL_SUITE_ID_PSK_WITH_CAMELLIA_256_GCM_SHA384             0xC08F
#define SSL_SUITE_ID_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256         0xC090
#define SSL_SUITE_ID_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384         0xC091
#define SSL_SUITE_ID_RSA_PSK_WITH_CAMELLIA_128_GCM_SHA256         0xC092
#define SSL_SUITE_ID_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384         0xC093
#define SSL_SUITE_ID_PSK_WITH_CAMELLIA_128_CBC_SHA256             0xC094
#define SSL_SUITE_ID_PSK_WITH_CAMELLIA_256_CBC_SHA384             0xC095
#define SSL_SUITE_ID_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256         0xC096
#define SSL_SUITE_ID_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384         0xC097
#define SSL_SUITE_ID_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256         0xC098
#define SSL_SUITE_ID_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384         0xC099
#define SSL_SUITE_ID_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256       0xC09A
#define SSL_SUITE_ID_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384       0xC09B
#define SSL_SUITE_ID_RSA_WITH_AES_128_CCM                         0xC09C
#define SSL_SUITE_ID_RSA_WITH_AES_256_CCM                         0xC09D
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_128_CCM                     0xC09E
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_256_CCM                     0xC09F
#define SSL_SUITE_ID_RSA_WITH_AES_128_CCM_8                       0xC0A0
#define SSL_SUITE_ID_RSA_WITH_AES_256_CCM_8                       0xC0A1
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_128_CCM_8                   0xC0A2
#define SSL_SUITE_ID_DHE_RSA_WITH_AES_256_CCM_8                   0xC0A3
#define SSL_SUITE_ID_PSK_WITH_AES_128_CCM                         0xC0A4
#define SSL_SUITE_ID_PSK_WITH_AES_256_CCM                         0xC0A5
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_128_CCM                     0xC0A6
#define SSL_SUITE_ID_DHE_PSK_WITH_AES_256_CCM                     0xC0A7
#define SSL_SUITE_ID_PSK_WITH_AES_128_CCM_8                       0xC0A8
#define SSL_SUITE_ID_PSK_WITH_AES_256_CCM_8                       0xC0A9
#define SSL_SUITE_ID_PSK_DHE_WITH_AES_128_CCM_8                   0xC0AA
#define SSL_SUITE_ID_PSK_DHE_WITH_AES_256_CCM_8                   0xC0AB
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_128_CCM                 0xC0AC
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_256_CCM                 0xC0AD
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_128_CCM_8               0xC0AE
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_AES_256_CCM_8               0xC0AF
#define SSL_SUITE_ID_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256      0xCCA8
#define SSL_SUITE_ID_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256    0xCCA9
#define SSL_SUITE_ID_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256        0xCCAA
```

4.1.3 Session flags

Description

Flags set by the user or by emSSL.

Definition

```
#define SSL_SESSION_FLAG_REQUEST_CERTIFICATE                   0x0001
#define SSL_SESSION_FLAG_CERTIFICATE_RECEIVED                  0x0002
#define SSL_SESSION_FLAG_CERTIFICATE_REQUEST_RECEIVED          0x0004
#define SSL_SESSION_FLAG_CERTIFICATE_SENT                      0x0008
#define SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_ID             0x0010
#define SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_TICKET         0x0020
#define SSL_SESSION_FLAG_REQUEST_RESUME_CRITICAL               0x0040
#define SSL_SESSION_FLAG_RESUME_GRANTED                        0x0080
#define SSL_SESSION_FLAG_DISABLE_SESSION_TICKET                0x0100
#define SSL_SESSION_FLAG_SESSION_TICKET_EXPECTED               0x0200
#define SSL_SESSION_FLAG_SESSION_TICKET_EXTENSION_ADVERTISED   0x0400
#define SSL_SESSION_FLAG_SESSION_TICKET_RECEIVED               0x0800
#define SSL_SESSION_FLAG_DISABLE_RSA_PMS_VERSION_CHECK         0x1000
#define SSL_SESSION_FLAG_REQUEST_PREVENT_FALLBACK              0x2000
#define SSL_SESSION_FLAG_PREFER_SERVER_ORDER                   0x4000
#define SSL_SESSION_FLAG_REQUIRE_STRICT_TLS_CLOSE              0x8000
```

Symbols

Definition	Description
SSL_SESSION_FLAG_REQUEST_CERTIFICATE	Set by user: emSSL Server must request a client certificate for mutual authentication.
SSL_SESSION_FLAG_CERTIFICATE_RECEIVED	Set by emSSL: Client provided a valid certificate to emSSL server.
SSL_SESSION_FLAG_CERTIFICATE_REQUEST_RECEIVED	Set by emSSL: Server requested certificate, if we have one.
SSL_SESSION_FLAG_CERTIFICATE_SENT	Set by emSSL: Client sent a valid certificate as we have one.
SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_ID	Set by emSSL: Indicates session resumption is requested by session ID.
SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_TICKET	Set by emSSL: Indicates session resumption is requested by session ticket.
SSL_SESSION_FLAG_REQUEST_RESUME_CRITICAL	Set by user: If session resumption is requested, it is critical that a session is resumed (no new session offered). If not, fail.
SSL_SESSION_FLAG_RESUME_GRANTED	Set by emSSL: After connection completes, indicates that a session is successfully resumed.
SSL_SESSION_FLAG_DISABLE_SESSION_TICKET	Set by user: Do not use session ticket even if configured.
SSL_SESSION_FLAG_SESSION_TICKET_EXPECTED	Set by emSSL: The client knows the server supports session tickets.
SSL_SESSION_FLAG_SESSION_TICKET_EXTENSION_ADVERTISED	Set by emSSL: The server indicates its willingness to issue session tickets.
SSL_SESSION_FLAG_SESSION_TICKET_RECEIVED	Set by emSSL: The client has received a session ticket.
SSL_SESSION_FLAG_DISABLE_RSA_PMS_VERSION_CHECK	Set by user: Disable TLS 1.0 version check [TLS1v2 https://tools.ietf.org/html/rfc5246#section-7.4.7.1].

Definition	Description
SSL_SESSION_FLAG_REQUEST_PREVENT_FALLBACK	Set by user: Request Fallback SCSV signaling cipher suite [RFC7507 https://tools.ietf.org/html/rfc7507].
SSL_SESSION_FLAG_PREFER_SERVER_ORDER	Set by user: Prefer the server's cipher suite order to the client's preferred order.
SSL_SESSION_FLAG_REQUIRE_STRICT_TLS_CLOSE	Set by user: Require a TLS close-notify from peer to close the session.

4.1.4 Logging flags

Description

Flags that control log output.

Definition

```
#define SSL_LOG_ERROR              (1uL <<  0)
#define SSL_LOG_RECORD             (1uL <<  1)
#define SSL_LOG_SIGNATURES         (1uL <<  2)
#define SSL_LOG_CERTIFICATES       (1uL <<  3)
#define SSL_LOG_VERIFY_DATA        (1uL <<  4)
#define SSL_LOG_STATES             (1uL <<  5)
#define SSL_LOG_KEYS               (1uL <<  6)
#define SSL_LOG_CIPHER             (1uL <<  7)
#define SSL_LOG_SOCKET_SEND        (1uL <<  8)
#define SSL_LOG_SOCKET_RECV        (1uL <<  9)
#define SSL_LOG_SUITES             (1uL << 10)
#define SSL_LOG_PRF                (1uL << 11)
#define SSL_LOG_HANDSHAKE          (1uL << 12)
#define SSL_LOG_MESSAGES           (1uL << 13)
#define SSL_LOG_CONFIG             (1uL << 14)
#define SSL_LOG_ALERT              (1uL << 15)
#define SSL_LOG_APP                (1uL << 31)
```

Symbols

Definition	Description
SSL_LOG_ERROR	Log all error status returns generated by emSSL.
SSL_LOG_RECORD	Log record-layer protocol details.
SSL_LOG_SIGNATURES	Log signature operations.
SSL_LOG_CERTIFICATES	Log certificate-related information; usually used in conjunction with SSL_LOG_SIGNATURES.
SSL_LOG_VERIFY_DATA	Log verification data; usually used in conjunction with SSL_LOG_HANDSHAKE.
SSL_LOG_STATES	Log SSL state machine transitions.
SSL_LOG_KEYS	Log key derivation for session bulk encryption keys; usually used in conjunction with SSL_LOG_CRYPTO.
SSL_LOG_CIPHER	Log cipher encryption and decryption.
SSL_LOG_SOCKET_SEND	Log raw data sent over an SSL connection.
SSL_LOG_SOCKET_RECV	Log raw data received over an SSL connection.
SSL_LOG_SUITES	Log agreed SSL cipher suite.
SSL_LOG_PRF	Log inputs and outputs of the SSL PRF function.
SSL_LOG_HANDSHAKE	Log data contributing to the SSL handshake when computing verification data.
SSL_LOG_MESSAGES	Log decoded SSL messages.
SSL_LOG_CONFIG	Log emSSL configuration on startup.
SSL_LOG_ALERT	Log received alert messages.
SSL_LOG_APP	Log application messages.

Additional information

Flags are added using `SSL_AddLogFilter()` and removed using `SSL_RemoveLogFilter()`.

4.1.5 Warning flags

Description

Flags that control warning output.

Definition

```
#define SSL_WARN_CRYPTO     (1uL << 0)
#define SSL_WARN_IGNORE     (1uL << 1)
#define SSL_WARN_X509       (1uL << 2)
#define SSL_WARN_CONFIG     (1uL << 3)
#define SSL_WARN_TICKETS    (1uL << 4)
```

Symbols

Definition	Description
SSL_WARN_CRYPTO	Warn on cryptography-related errors such as bad message formatting or bad key parameters.
SSL_WARN_IGNORE	Warn on purposely-ignored nonfatal conditions, such as SSL extensions that are not recognized by the current emSSL implementation which allow forward compatibility with TLS specifications.
SSL_WARN_X509	Warn on nonfatal X.509 certificate issues identified by emSSL which allow forward compatibility with new X.509 capabilities.
SSL_WARN_CONFIG	Warn on configuration issues on startup.
SSL_WARN_TICKETS	Warn on session ticket problems.

Additional information

Flags are added using `SSL_AddWarnFilter()` and removed using `SSL_RemoveWarnFilter()`.

4.2 Information functions

The table below lists the functions that return emSSL information.

Function	Description
`SSL_GetVersionText()`	Get emSSL version as printable string.
`SSL_GetCopyrightText()`	Get emSSL copyright as printable string.

4.2.1 SSL_GetVersionText()

Description

Get emSSL version as printable string.

Prototype

```
char *SSL_GetVersionText(void);
```

Return value

Zero-terminated version string.

4.2.2 SSL_GetCopyrightText()

Description

Get emSSL copyright as printable string.

Prototype

```
char *SSL_GetCopyrightText(void);
```

Return value

Zero-terminated copyright string.

4.3 Control functions

The table below lists the functions provided by the emSSL API. Detailed description of each function is found in the sections that follow.

Function	Description
`SSL_Exit()`	Finalize the SSL module.
`SSL_Init()`	Initialize the SSL module.

4.3.1 SSL_Exit()

Description

Finalize the SSL module.

Prototype

```
void SSL_Exit(void);
```

Additional information

This function deinitializes the SSL module. Once finalized, no further calls must be made to the emSSL API.

4.3.2 SSL_Init()

Description

Initialize the SSL module.

Prototype

```
void SSL_Init(void);
```

Additional information

Before using an SSL service, you must call `SSL_Init`. As part of SSL initialization, emSSL ensures that the shared CRYPTO component is initialized.

4.4 Configuration functions

The table below lists the functions that configure emSSL for operation.

Function	Description
SSL_CIPHER_Add()	Add cipher to emSSL.
SSL_CLIENT_ConfigMutualAuth()	Support mutual authentication, client mode.
SSL_CURVE_Add()	Add elliptic curve to emSSL.
SSL_CURVE_Remove()	Remove an elliptic curve.
SSL_CURVE_GetName()	Return the standard name for an elliptic curve.
SSL_MAC_Add()	Add MAC support to emSSL.
SSL_MEM_Add()	Add memory to emSSL.
SSL_MEM_ConfigSystem()	Configure the SSL memory allocator to use the C system heap.
SSL_MEM_GetContext()	Get the default memory allocation context for the SSL module.
SSL_PROTOCOL_Add()	Add TLS protocol to emSSL.
SSL_PROTOCOL_GetText()	Decode the TLS protocol to a textual representation.
SSL_ROOT_CERTIFICATE_Add()	Add root certificate to emSSL.
SSL_SetDefaultCertificateAPI()	Set default certificate API used by all SSL connections.
SSL_SERVER_ConfigMutualAuth()	Support mutual authentication, server mode.
SSL_SIGNATURE_ALGORITHM_Add()	Add signature algorithm to emSSL.
SSL_SIGNATURE_SIGN_Add()	Add signature signer to emSSL.
SSL_SIGNATURE_VERIFY_Add()	Add signature verifier to emSSL.
SSL_SUITE_Add()	Add cipher suite to emSSL.

4.4.1 SSL_CIPHER_Add()

Description

Add cipher to emSSL.

Prototype

```
void SSL_CIPHER_Add(const SSL_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Cipher API to add.

Additional information

Adds a bulk cipher to emSSL in support of cipher suites. This function must only be called during emSSL configuration.

Implemented ciphers

The following ciphers are provided by emSSL:

```
extern const SSL_CIPHER_API SSL_CIPHER_AES_128_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_256_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_128_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_256_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_128_CCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_256_CCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_128_CCM_8_API;
extern const SSL_CIPHER_API SSL_CIPHER_AES_256_CCM_8_API;
extern const SSL_CIPHER_API SSL_CIPHER_ARIA_128_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_ARIA_256_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_ARIA_128_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_ARIA_256_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_CAMELLIA_128_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_CAMELLIA_256_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_CAMELLIA_128_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_CAMELLIA_256_GCM_API;
extern const SSL_CIPHER_API SSL_CIPHER_SEED_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_DES_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_3DES_EDE_CBC_API;
extern const SSL_CIPHER_API SSL_CIPHER_RC4_128_API;
extern const SSL_CIPHER_API SSL_CIPHER_CHACHA20_POLY1305_API;
```

See also

Adding ciphers on page 135.

4.4.2 SSL_CLIENT_ConfigMutualAuth()

Description

Support mutual authentication, client mode.

Prototype

```
void SSL_CLIENT_ConfigMutualAuth(void);
```

Additional information

This function adds support for mutual authentication for connections operating in client mode, i.e. to answer with a client certificate when the server requests one.

4.4.3 SSL_CURVE_Add()

Description

Add elliptic curve to emSSL.

Prototype

```
void SSL_CURVE_Add(const SSL_CURVE * pCurve);
```

Parameters

Parameter	Description
pCurve	Pointer to curve.

Additional information

Adds a single elliptic curve that will be offered by the server when using elliptic curve suites.

When agreeing cipher suites and parsing ECDSA certificates, both sides must support a common elliptic curve which defines the appropriate security. When parsing ECDSA certificates, you must ensure that appropriate elliptic curves are registered in order to extract and use the enclosed public keys.

Implemented curves

The following curves are provided by emSSL:

```
extern const SSL_CURVE SSL_CURVE_secp192k1;
extern const SSL_CURVE SSL_CURVE_secp192r1;
extern const SSL_CURVE SSL_CURVE_secp224k1;
extern const SSL_CURVE SSL_CURVE_secp224r1;
extern const SSL_CURVE SSL_CURVE_secp256k1;
extern const SSL_CURVE SSL_CURVE_secp256r1;
extern const SSL_CURVE SSL_CURVE_secp384r1;
extern const SSL_CURVE SSL_CURVE_secp521r1;
extern const SSL_CURVE SSL_CURVE_brainpoolP256r1;
extern const SSL_CURVE SSL_CURVE_brainpoolP384r1;
extern const SSL_CURVE SSL_CURVE_brainpoolP512r1;
extern const SSL_CURVE SSL_CURVE_Curve25519;
```

Although the underlying crypgraphic algorithm library offers more curves, the SSL protocol only supports a subset of all standardized elliptic curves.

See also

Adding elliptic curves on page 138.

4.4.4 SSL_CURVE_Remove()

Description

Remove an elliptic curve.

Prototype

```
void SSL_CURVE_Remove(const SSL_CURVE * pCurve);
```

Parameters

Parameter	Description
pCurve	Pointer to curve.

Additional information

Removes a single elliptic curve from the list of supported curves.

4.4.5 SSL_CURVE_GetName()

Description

Return the standard name for an elliptic curve.

Prototype

```
char *SSL_CURVE_GetName(unsigned ID);
```

Parameters

Parameter	Description
ID	SSL named curve ID.

Return value

Curve name or “UNKNOWN” if the curve ID is not known.

4.4.6 SSL_MAC_Add()

Description

Add MAC support to emSSL.

Prototype

```
void SSL_MAC_Add(const SSL_MAC_API * pAPI);
```

Parameters

Parameter	Description
pAPI	MAC API to add.

Additional information

Adds a hash/HMAC algorithm to emSSL in support of cipher suites.

Implemented MACs

The following hash/HMAC algorithms are provided by emSSL:

```
extern const SSL_MAC_API SSL_MAC_MD5_API;
extern const SSL_MAC_API SSL_MAC_SHA_API;
extern const SSL_MAC_API SSL_MAC_SHA224_API;
extern const SSL_MAC_API SSL_MAC_SHA256_API;
extern const SSL_MAC_API SSL_MAC_SHA384_API;
extern const SSL_MAC_API SSL_MAC_SHA512_API;
```

See also

Adding MACs on page 136.

4.4.7 SSL_MEM_Add()

Description

Add memory to emSSL.

Prototype

```
void SSL_MEM_Add(void     * pStore,
                 unsigned   NumBytesStore);
```

Parameters

Parameter	Description
pStore	Pointer to the first byte of memory to be added. This must be correctly aligned for the processor and compiler combination.
NumBytesStore	Number of bytes in memory block.

Additional information

This function must be called a maximum of one time to add memory to emSSL. Once the memory is added, the heap implementation that manages it is selected. If emSSL is to use the C system heap, this function should not be called.

See also

RAM use on page 233.

4.4.8 SSL_MEM_ConfigSystem()

Description

Configure the SSL memory allocator to use the C system heap.

Prototype

```
void SSL_MEM_ConfigSystem(void);
```

Additional information

This function sets emSSL's allocator to utilize the C system heap through calls to malloc(), free(), and realloc(). For workstation-class machines or PCs, where memory is plentiful, this allocator suffices.

If emSSL uses the C system heap, `SSL_MEM_Add()` must not be called.

4.4.9 SSL_MEM_GetContext()

Description

Get the default memory allocation context for the SSL module.

Prototype

```
void SSL_MEM_GetContext(SEGGER_MEM_CONTEXT ** pMem);
```

Parameters

Parameter	Description
pMem	Assigned pointer to default memory context.

4.4.10 SSL_PROTOCOL_Add()

Description

Add TLS protocol to emSSL.

Prototype

```
void SSL_PROTOCOL_Add(const SSL_PROTOCOL_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Protocol API to add.

Additional information

Adds a single TLS protocol to emSSL.

Implemented protocols

The following protocols are provided by emSSL:

```
extern const SSL_PROTOCOL_API SSL_PROTOCOL_TLS1v0_API;
extern const SSL_PROTOCOL_API SSL_PROTOCOL_TLS1v1_API;
extern const SSL_PROTOCOL_API SSL_PROTOCOL_TLS1v2_API;
```

See also

Adding TLS protocols on page 138

4.4.11 SSL_PROTOCOL_GetText()

Description

Decode the TLS protocol to a textual representation.

Prototype

```
char *SSL_PROTOCOL_GetText(U16 Protocol);
```

Parameters

Parameter	Description
`Protocol`	TLS protocol ID.

Return value

Nonzero pointer to the protocol name.

4.4.12 SSL_ROOT_CERTIFICATE_Add()

Description

Add root certificate to emSSL.

Prototype

```
void SSL_ROOT_CERTIFICATE_Add(SSL_ROOT_CERTIFICATE * pCert);
```

Parameters

Parameter	Description
pCert	Pointer to certificate to add as a trusted root.

See also

Installing root certificates on page 49.

4.4.13 SSL_SERVER_ConfigMutualAuth()

Description

Support mutual authentication, server mode.

Prototype

```
void SSL_SERVER_ConfigMutualAuth(void);
```

Additional information

This function adds support for mutual authentication for connections operating in server mode, i.e. to ask the for a certificate when the server requires one from the client.

Note that you must add server mutual authentication support if you intend to set the flag `SSL_SESSION_FLAG_REQUEST_CERTIFICATE` in server mode.

See also

SSL_SESSION_SetFlags on page 101.

4.4.14 SSL_SetDefaultCertificateAPI()

Description

Set default certificate API used by all SSL connections.

Prototype

```
void SSL_SetDefaultCertificateAPI(const SSL_CERTIFICATE_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to certificate API that will be used by default for new connections.

Additional information

This function sets the default certificate API to use when establishing a connection. The default certificate API can be overridden on a per-session basis using the function `SSL_SESSION_SetCertificateAPI()`.

4.4.15 SSL_SIGNATURE_ALGORITHM_Add()

Description

Add signature algorithm to emSSL.

Prototype

```
void SSL_SIGNATURE_ALGORITHM_Add(unsigned ID);
```

Parameters

Parameter	Description
ID	Signature algorithm to add.

Additional information

Adds a signature algorithm to emSSL in order to advertise it when negotiating a connection.

Implemented signature algorithms

The following signature algorithms are provided by emSSL:

RSA

```
SSL_SIGNATURE_MD5_WITH_RSA_ENCRYPTION
SSL_SIGNATURE_SHA_WITH_RSA_ENCRYPTION
SSL_SIGNATURE_SHA224_WITH_RSA_ENCRYPTION
SSL_SIGNATURE_SHA256_WITH_RSA_ENCRYPTION
SSL_SIGNATURE_SHA384_WITH_RSA_ENCRYPTION
SSL_SIGNATURE_SHA512_WITH_RSA_ENCRYPTION
```

DSA

```
SSL_SIGNATURE_SHA_WITH_DSA
```

ECDSA

```
SSL_SIGNATURE_SHA_WITH_ECDSA
SSL_SIGNATURE_SHA224_WITH_ECDSA
SSL_SIGNATURE_SHA256_WITH_ECDSA
SSL_SIGNATURE_SHA384_WITH_ECDSA
SSL_SIGNATURE_SHA512_WITH_ECDSA
```

See also

Adding signature algorithms on page 137.

4.4.16 SSL_SIGNATURE_SIGN_Add()

Description

Add signature signer to emSSL.

Prototype

```
void SSL_SIGNATURE_SIGN_Add(const SSL_SIGNATURE_SIGN_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Signature signer API to add.

Additional information

Adds a signature signer to emSSL in support of cipher suites.

Implemented message signers

The following message signers are provided by emSSL:

```
extern const SSL_SIGNATURE_SIGN_API SSL_SIGNATURE_SIGN_RSA_API;
extern const SSL_SIGNATURE_SIGN_API SSL_SIGNATURE_SIGN_ECDSA_API;
```

See also

Adding public key message signers on page 136.

4.4.17 SSL_SIGNATURE_VERIFY_Add()

Description

Add signature verifier to emSSL.

Prototype

```
void SSL_SIGNATURE_VERIFY_Add(const SSL_SIGNATURE_VERIFY_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Signature verification API to add.

Additional information

Adds a signature verifier to emSSL in support of cipher suites.

Implemented signature verifiers

The following signature verifiers are provided by emSSL:

```
extern const SSL_SIGNATURE_VERIFY_API SSL_SIGNATURE_VERIFY_RSA_API;
extern const SSL_SIGNATURE_VERIFY_API SSL_SIGNATURE_VERIFY_DSA_API;
extern const SSL_SIGNATURE_VERIFY_API SSL_SIGNATURE_VERIFY_ECDSA_API;
```

See also

Adding public key signature verifiers on page 136.

4.4.18 SSL_SUITE_Add()

Description

Add cipher suite to emSSL.

Prototype

```
void SSL_SUITE_Add(const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pSuite	Pointer to cipher suite to add.

Additional information

The order in which suites are added defines the default preference order of suites in the negotiation phase of a client connection..

Implemented cipher suites

The following cipher suites are provided by emSSL:

```
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_NULL_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CCM;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CCM;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CCM_8;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CCM_8;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_NULL_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_NULL_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_NULL_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_256_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_128_CCM;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_256_CCM;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_128_CCM_8;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_AES_256_CCM_8;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_RC4_128_MD5;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_256_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_128_CCM;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_256_CCM;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_128_CCM_8;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_AES_256_CCM_8;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_ARIA_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_256_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_RSA_WITH_CAMELLIA_256_GCM_SHA384;
```

See also

Adding cipher suites on page 132.

4.5 Session control functions

The table below lists the functions that are used for SSL sessions.

Function	Description
`SSL_SESSION_Accept()`	Negotiate an SSL connection as a server.
`SSL_SESSION_ClrFlags()`	Clear session-related flags.
`SSL_SESSION_Connect()`	Connect to a server.
`SSL_SESSION_Disconnect()`	Disconnect a client or server SSL connection.
`SSL_SESSION_GetSuite()`	Retrieve the active suite for an SSL session.
`SSL_SESSION_Prepare()`	Prepare SSL session before connection.
`SSL_SESSION_Receive()`	Receive data over an established SSL/TLS connection.
`SSL_SESSION_QueryFlags()`	Query session-related flags.
`SSL_SESSION_Send()`	Send data over an established SSL/TLS connection.
`SSL_SESSION_SendStr()`	Send null-terminated string to peer.
`SSL_SESSION_SetAllowedSuites()`	Set the cipher suites to offer or accept for all connections.
`SSL_SESSION_SetFlags()`	Set session-related flags.
`SSL_SESSION_SetCertificateAPI()`	Set the API to use to handle PKI certificates and keys.
`SSL_SESSION_SetProtocolRange()`	Set the protocol range supported by the SSL connection.

4.5.1 SSL_SESSION_Accept()

Description

Negotiate an SSL connection as a server.

Prototype

```
int SSL_SESSION_Accept(SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.

Return value

< 0	Processing error.
≥ 0	Success.

Additional information

This function attempts to negotiate an SSL connection with the SSL client. To negotiate a connection for a specific SSL version, restrict the range of supported TLS versions using SSL_SESSION_SetProtocolRange.

See also

SSL_SESSION_SetProtocolRange on page 104.

4.5.2 SSL_SESSION_ClrFlags()

Description

Clear session-related flags.

Prototype

```
void SSL_SESSION_ClrFlags(SSL_SESSION * pSelf,
                          unsigned      Flags);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
Flags	Bitwise-or of flags to clear.

Additional information

This function clears session-related flags, for instance requesting that the client provides its certificate during negotiation.

This function must be called after initializing the session using SSL_SESSION_Prepare() and before making or accepting connections using SSL_SESSION_Connect() or SSL_SESSION_Accept().

4.5.3 SSL_SESSION_Connect()

Description

Connect to a server.

Prototype

```
int SSL_SESSION_Connect(      SSL_SESSION * pSelf,
                        const char        * sServerName);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
sServerName	Name of server we wish to connect to.

Return value

≥ 0	Success.
< 0	Processing error.

Additional information

This function attempts to negotiate an SSL connection to the SSL server whose name is sServerName. The server name is used in the Server Name Indication extension of RFC 6066. If sServerName is the null pointer, the Server Name Indication extension is not included in the SSL handshake.

To negotiate a client connection for a specific SSL version, restrict the range of supported TLS versions using SSL_SESSION_SetProtocolRange.

See also

SSL_SESSION_SetProtocolRange on page 104.

4.5.4 SSL_SESSION_Disconnect()

Description

Disconnect a client or server SSL connection..

Prototype

```
void SSL_SESSION_Disconnect(SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.

Additional information

This function disconnects an existing client or server SSL connection with the SSL client. After disconnection, you must not attempt to send or receive data over the connection.

4.5.5 SSL_SESSION_GetSuite()

Description

Retrieve the active suite for an SSL session.

Prototype

```
SSL_SUITE *SSL_SESSION_GetSuite(SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.

Return value

Cipher suite in use (can be null if no suite is agreed).

Additional information

It is acceptable to use this function during a certificate callback in order to determine the type of certificate to return to emSSL.

4.5.6 SSL_SESSION_Prepare()

Description

Prepare SSL session before connection.

Prototype

```
void SSL_SESSION_Prepare(      SSL_SESSION       * pSelf,
                               int                 Socket,
                         const SSL_TRANSPORT_API * pAPI);
```

Parameters

Parameter	Description
pSelf	Session context to prepare.
Socket	Socket ID used by the transport API.
pAPI	Pointer to the transport API to use for the connection.

Additional information

This function initializes the SSL session and sets the communications API to use in order to transport SSL messages.

By default the session is initialized to support TLS 1.0 through 1.2. You can customize the connection protocol version for the connection using SSL_SESSION_SetProtocolRange before establishing a client or server connection.

See also

SSL_SESSION_SetProtocolRange on page 104.

4.5.7 SSL_SESSION_QueryFlags()

Description

Query session-related flags.

Prototype

```
unsigned SSL_SESSION_QueryFlags(SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.

Return value

Bitwise-or of the flags set for the session.

Additional information

This function retrieves flags set during the lifetime of a session.

The following flags are defined for an SSL server:

Flag	Description
SSL_SESSION_FLAG_REQUEST_CERTIFICATE	SSL server requests a certificate from the client during connection setup for mutual authentication.
SSL_SESSION_FLAG_RECEIVED_CERTIFI-CATE	SSL server received a valid certificate from the client during connection setup.
SSL_SESSION_FLAG_REQUEST_PREVEN-T_FALLBACK	SSL client adds the Fallback signalling sipher suite to the Client Hello message (if required) to prevent version rollback.

4.5.8 SSL_SESSION_Receive()

Description

Receive data over an established SSL/TLS connection.

Prototype

```
int SSL_SESSION_Receive(SSL_SESSION * pSelf,
                        void        * pData,
                        unsigned      DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
pData	Pointer to destination that will receive TLS data.
DataLen	Size of destination array in bytes.

Return value

< 0	Processing error.
≥ 0	Number of bytes received successfully.

Additional information

This function waits for data to arrive on the established TLS connection before returning. The number of bytes returned will lie between zero and DataLen. Any alerts sent by the peer are processed and converted into appropriate status codes in the return value.

4.5.9 SSL_SESSION_Send()

Description

Send data over an established SSL/TLS connection.

Prototype

```
int SSL_SESSION_Send(      SSL_SESSION * pSelf,
                     const void        * pData,
                           unsigned      DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
pData	Pointer to octet string to send over SSL.
DataLen	Octet length of the octet string to send.

Return value

< 0	Processing error.
≥ 0	Number of bytes send from the octet string.

Additional information

This function sends the data over the established SSL connection using the negotiated cipher suite and protocol version. Any alerts sent by the peer are processed and converted into appropriate status codes in the return value.

The data to send is immediately encrypted and sent to the peer—emSSL does not perform any buffering to opportunistically combine TLS packets.

Application data will be fragmented, as required, across multiple protocol packets according to the setting of `SSL_MAX_APP_DATA_FRAGMENT_LEN`.

4.5.10 SSL_SESSION_SendStr()

Description

Send null-terminated string to peer.

Prototype

```
int SSL_SESSION_SendStr(      SSL_SESSION * pSelf,
                        const char        * sText);
```

Parameters

Parameter	Description
pSelf	Pointer to session context to send over.
sText	Null-terminated string to send.

Return value

< 0	Processing error.
≥ 0	Success.

Additional information

This function is a convenience that wraps a call to SSL_SESSION_Send() to send the null-terminated string over an established SSL connection.

See also

SSL_SESSION_Send on page 98.

4.5.11 SSL_SESSION_SetAllowedSuites()

Description

Set the cipher suites to offer or accept for all connections.

Prototype

```
void SSL_SESSION_SetAllowedSuites(      SSL_SESSION * pSelf,
                                  const U16         * pSuites,
                                        unsigned      SuiteCnt);
```

Parameters

Parameter	Description
pSelf	Session context to override.
pSuites	Pointer to array of cipher suite IDs to allow.
SuiteCnt	Count of cipher suite IDs in the array.

Additional information

This function sets the cipher suites that are to be offered by an SSL client and supported by an SSL server for a specific connection. This function enables restriction of cipher suites on a per-session basis such that client and server connections may vary the set of suites that are offered or accepted.

This function must be called after initializing the session using `SSL_SESSION_Prepare()` and before making or accepting connections using `SSL_SESSION_Connect()` or `SSL_SESSION_Accept()`.

4.5.12 SSL_SESSION_SetFlags()

Description

Set session-related flags.

Prototype

```
void SSL_SESSION_SetFlags(SSL_SESSION * pSelf,
                          unsigned      Flags);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
Flags	Bitwise-or of flags to set.

Additional information

This function sets session-related flags, for instance requesting that the client provides its certificate during negotiation.

This function must be called after initializing the session using `SSL_SESSION_Prepare()` and before making or accepting connections using `SSL_SESSION_Connect()` or `SSL_SESSION_Accept()`.

Additional information

The following flags are defined for an SSL server:

Flag	Description
SSL_SESSION_FLAG_REQUEST_CERTIFICATE	SSL server requests a certificate from the client during connection setup for mutual authentication.
SSL_SESSION_FLAG_REQUEST_RESUME_CRITICAL	When connecting to a server with session resumption, indicate that it is critical the session is resumed rather than a new session created. If the session is not resumed, the connection is failed.
SSL_SESSION_FLAG_DISABLE_SESSION_TICKET	Disable resumption by session ticket and any advertising of session ticket capability.
SSL_SESSION_FLAG_DISABLE_RSA_PMS_VERSION_CHECK	Disable TLS 1.0 premaster secret version check. See RFC 5246 section 7.4.7.1.
SSL_SESSION_FLAG_REQUIRE_STRICT_TLS_CLOSE	It is an error in the TLS protocol if a socket is closed at the TCP layer without the peer preceding the closure with a close-notify alert to terminate the TLS session. By default and to provide maximum compatibility with non-compliant TLS stacks and applications, emSSL will tolerate graceful closure of TCP sockets and propagate that graceful closure to the TLS layer and also terminate the TLS session gracefully. It does this only for graceful closure of the socket: if the socket is shut down with an error, that error is propagated to the TLS layer regardless. Setting this flag requires that a close-notify be sent to close the SSL session, and socket closure at the TCP lay-

Flag	Description
	er without the close-notify alert is consid-ered an error.

4.5.13 SSL_SESSION_SetCertificateAPI()

Description

Set the API to use to handle PKI certificates and keys.

Prototype

```
void SSL_SESSION_SetCertificateAPI(      SSL_SESSION         * pSelf,
                                   const SSL_CERTIFICATE_API * pAPI);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
pAPI	Pointer to certificate API that will be used for the session.

Additional information

This function sets the per-session API for validation of presented server certificates, for retrieving SSL server certificates and private keys (for server connections) and client certificates and keys (for client connections).

Installed certificates are required for all server connections other than those with anonymous cipher suites (which are vulnerable to man-in-the-middle attacks). It is optional for clients: if clients wish to offer certificates for mutual authentication, they must implement the API.

This function must be called after initializing the session using `SSL_SESSION_Prepare()` and before making or accepting connections using `SSL_SESSION_Connect()` or `SSL_SESSION_Accept()`.

4.5.14 SSL_SESSION_SetProtocolRange()

Description

Set the protocol range supported by the SSL connection.

Prototype

```
void SSL_SESSION_SetProtocolRange(SSL_SESSION * pSelf,
                                  U16           MinVersion,
                                  U16           MaxVersion);
```

Parameters

Parameter	Description
pSelf	Pointer to session context.
MinVersion	Minimum SSL/TLS version supported.
MaxVersion	Maximum SSL/TLS version supported.

Additional information

By default, a new SSL session is initialized to offer support for all TLS versions installed during initialization. If you wish to restrict the protocols offered by a server or required by a client within the configured range, you must call this function after initialization and before any connection attempt by client or server.

It is not possible to use this function to configure emSSL for protocols prior to TLS 1.0 because SSL 2 and SSL 3 are now considered insecure and emSSL offers no support for these protocols.

Example

This example limits the protocol to TLS 1.1 through TLS 1.2, thereby excluding connections using TLS 1.0 and previous versions.

```
static SSL_SESSION _Session;
//
SSL_SESSION_Prepare(&Session, Socket, &_IP_Transport);
SSL_SESSION_SetProtocolRange(&_Session, SSL_PROTOCOL_TLS_1v1, SSL_PROTOCOL_TLS_1v2);
```

4.6 Cipher suite functions

The table below lists the functions that query cipher suite configuration.

Function	Description
SSL_SUITE_CopyName()	Copy IANA name of the cipher suite.
SSL_SUITE_FindByID()	Find an installed cipher suite by IANA ID.
SSL_SUITE_FindByIndex()	Find an installed cipher suite by preference index.
SSL_SUITE_GetCipherName()	Get the cipher name for a cipher ID.
SSL_SUITE_GetIANASuiteName()	Get IANA name of a SSL suite ID.
SSL_SUITE_GetID()	Get the IANA cipher suite ID.
SSL_SUITE_GetKeyExchangeName()	Get the name of a key exchange ID.
SSL_SUITE_GetMACAlgorithmName()	Get the algorithm name for a MAC algorithm ID.
SSL_SUITE_GetPKAlgorithmName()	Get the name of a public key algorithm ID.
SSL_SUITE_QueryNull()	Does cipher suite have a null component?
SSL_SUITE_QueryPKAlgorithm()	Query the public key algorithm associated with a key exchange.
SSL_SUITE_QueryRequiresECC()	Does cipher suite require ECC support?
SSL_SUITE_QueryRequiresPSK()	Does cipher suite require PSK support?
SSL_SUITE_QueryValidity()	Is cipher suite supported and valid for a given TLS version?

4.6.1 SSL_SUITE_CopyName()

Description

Copy IANA name of the cipher suite.

Prototype

```
void SSL_SUITE_CopyName(      char      * pText,
                        const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pText	Pointer to a buffer of at least 80 characters to receive the name.
pSuite	The suite to return the IANA name of.

Additional information

The name has no leading “TLS” prefix, it starts with the key agreement scheme.

4.6.2 SSL_SUITE_FindByID()

Description

Find an installed cipher suite by IANA `ID`.

Prototype

```
SSL_SUITE *SSL_SUITE_FindByID(unsigned ID);
```

Parameters

Parameter	Description
`ID`	IANA cipher suite `ID`.

Return value

= 0	Cipher suite not found.
≠ 0	Cipher suite corresponding to the IANA `ID`.

Additional information

Only cipher suites installed by `SSL_SUITE_Add()` will be searched.

4.6.3 SSL_SUITE_FindByIndex()

Description

Find an installed cipher suite by preference index.

Prototype

```
SSL_SUITE *SSL_SUITE_FindByIndex(unsigned Index);
```

Parameters

Parameter	Description
Index	Preference index with zero indicating the first suite that was added.

Return value

= 0	Cipher suite not found.
≠ 0	Cipher suite corresponding to the preference index.

Additional information

Only cipher suites installed by SSL_SUITE_Add() will be iterated over. The suite that is added first has index 0, the second added has index 1, and so on.

4.6.4 SSL_SUITE_GetCipherName()

Description

Get the cipher name for a cipher ID.

Prototype

```
char *SSL_SUITE_GetCipherName(SSL_CIPHER_ID ID);
```

Parameters

Parameter	Description
ID	Cipher ID.

Return value

Zero-terminated string describing ID.

4.6.5 SSL_SUITE_GetIANASuiteName()

Description

Get IANA name of a SSL suite ID.

Prototype

```
char *SSL_SUITE_GetIANASuiteName(unsigned ID);
```

Parameters

Parameter	Description
ID	TLS suite ID.

Return value

The corresponding suite name or “Unknown” if the suite ID is not known.

4.6.6 SSL_SUITE_GetID()

Description

Get the IANA cipher suite ID.

Prototype

```
int SSL_SUITE_GetID(const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pSuite	The suite under consideration.

Return value

IANA cipher suite ID.

Additional information

The list of cipher suites defined by IANA is maintained here:

http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

4.6.7 SSL_SUITE_GetKeyExchangeName()

Description

Get the name of a key exchange ID.

Prototype

```
char *SSL_SUITE_GetKeyExchangeName(SSL_KEY_EXCHANGE_ID ID);
```

Parameters

Parameter	Description
ID	Key exchange ID.

Return value

Zero-terminated string describing ID.

4.6.8 SSL_SUITE_GetMACAlgorithmName()

Description

Get the algorithm name for a MAC algorithm ID.

Prototype

```
char *SSL_SUITE_GetMACAlgorithmName(SSL_HASH_ALGORITHM_ID ID);
```

Parameters

Parameter	Description
ID	MAC algorithm ID.

Return value

Zero-terminated string describing ID.

4.6.9 SSL_SUITE_GetPKAlgorithmName()

Description

Get the name of a public key algorithm ID.

Prototype

```
char *SSL_SUITE_GetPKAlgorithmName(CRYPTO_X509_PK_ALGORITHM_ID ID);
```

Parameters

Parameter	Description
ID	Public key algorithm ID.

Return value

Zero-terminated string describing ID.

4.6.10 SSL_SUITE_QueryNull()

Description

Does cipher suite have a null component?

Prototype

```
int SSL_SUITE_QueryNull(const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pSuite	Suite under consideration.

Return value

= 0	No component of the suite is `NULL`.
≠ 0	Some component of the suite is `NULL`.

4.6.11 SSL_SUITE_QueryPKAlgorithm()

Description

Query the public key algorithm associated with a key exchange.

Prototype

```
CRYPTO_X509_PK_ALGORITHM_ID SSL_SUITE_QueryPKAlgorithm(SSL_KEY_EXCHANGE_ID ID);
```

Parameters

Parameter	Description
`ID`	Key exchange `ID`.

Return value

Public key algorithm associated with the key exchange mechanism `ID`.

4.6.12 SSL_SUITE_QueryRequiresECC()

Description

Does cipher suite require ECC support?

Prototype

```
int SSL_SUITE_QueryRequiresECC(const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pSuite	Suite under consideration.

Return value

= 0	Does not require ECC.
≠ 0	Requires ECC.

Additional information

Cipher suites that are ECDH or ECDHE will return true; others will returns false.

4.6.13 SSL_SUITE_QueryRequiresPSK()

Description

Does cipher suite require PSK support?

Prototype

```
int SSL_SUITE_QueryRequiresPSK(const SSL_SUITE * pSuite);
```

Parameters

Parameter	Description
pSuite	Suite under consideration.

Return value

= 0	Does not require PSK.
≠ 0	Requires PSK.

4.6.14 SSL_SUITE_QueryValidity()

Description

Is cipher suite supported and valid for a given TLS version?

Prototype

```
int SSL_SUITE_QueryValidity(const SSL_SUITE * pSuite,
                                  unsigned    Version);
```

Parameters

Parameter	Description
`pSuite`	Suite under consideration.
`Version`	Protocol ID of version of SSL/TLS to test.

Return value

= 0	Cipher suite is not valid for TLS version `Version`.
≠ 0	Cipher suite is valid for TLS version `Version`.

Additional information

Only the protocol versions defined by emSSL are valid which are:

- `SSL_PROTOCOL_ID_SSL_3v0`
- `SSL_PROTOCOL_ID_TLS_1v0`
- `SSL_PROTOCOL_ID_TLS_1v1`
- `SSL_PROTOCOL_ID_TLS_1v2`

4.7 Diagnostic functions

The table below lists the diagnostic functions provided by the emSSL API.

Function	Description
`SSL_AddLogFilter()`	Add filters to the active log filter.
`SSL_AddWarnFilter()`	Add filters to the active warning filter.
`SSL_ERROR_GetText()`	Decode an SSL error code.
`SSL_RemoveLogFilter()`	Remove filters from the active log filter.
`SSL_RemoveWarnFilter()`	Remove filters from the active warning filter.
`SSL_SetLogFilter()`	Set active log filter.
`SSL_SetWarnFilter()`	Set the active warning filter.

4.7.1 SSL_AddLogFilter()

Description

Add filters to the active log filter.

Prototype

```
U32 SSL_AddLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to enable.

Return value

The log filter mask before addition.

Additional information

This function adds to the existing log filter mask using a bitwise or of the new filter mask and the given filter mask.

Example

This example temporarily enables error logging during connection setup.

```
SSL_SESSION Session;
U32         Previous;
//
Previous = SSL_AddLogFilter(SSL_LOG_ERROR);
Status   = SSL_SESSION_Connect(&Session, "www.segger.com");
SSL_SetLogFilter(Previous);
```

See also

Logging flags on page 60, *`SSL_RemoveLogFilter`* on page 124.

4.7.2 SSL_AddWarnFilter()

Description

Add filters to the active warning filter.

Prototype

```
U32 SSL_AddWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to enable.

Return value

The warning filter mask before addition.

Additional information

This function adds to the existing warning filter mask using a bitwise or of the new filter mask and the given filter mask.

Example

This example temporarily enables alert warnings during connection setup.

```
SSL_SESSION Session;
U32         Previous;
//
Previous = SSL_AddWarnFilter(SSL_WARN_ALERT);
Status   = SSL_SESSION_Connect(&Session, "www.segger.com");
SSL_SetWarnFilter(Previous);
```

See also

- *SSL_SetWarnFilter* on page 127 for a list of warnings.
- *SSL_RemoveWarnFilter* on page 125.

4.7.3 SSL_ERROR_GetText()

Description

Decode an SSL error code.

Prototype

```
char *SSL_ERROR_GetText(int ErrorCode);
```

Parameters

Parameter	Description
ErrorCode	Error code returned by emSSL.

Return value

Zero-terminated string describing the emSSL error status.

4.7.4 SSL_RemoveLogFilter()

Description

Remove filters from the active log filter.

Prototype

```
U32 SSL_RemoveLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to disable.

Return value

The log filter mask before removal.

Additional information

This function removes the filters specified in `FilterMask` from the existing log filter.

Example

This example temporarily suspends error logging during connection setup. `SSL_SESSION Session;`

```
U32         Previous;
//
Previous = SSL_AddRemoveFilter(SSL_LOG_ERROR);
Status   = SSL_SESSION_Connect(&Session, "www.segger.com");
SSL_SetLogFilter(Previous);
```

See also

Logging flags on page 60, *`SSL_AddLogFilter`* on page 121.

4.7.5 SSL_RemoveWarnFilter()

Description

Remove filters from the active warning filter.

Prototype

```
U32 SSL_RemoveWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to disable.

Return value

The warning filter mask before removal.

Additional information

This function removes the filters specified in `FilterMask` from the existing log filter.

4.7.6 SSL_SetLogFilter()

Description

Set active log filter.

Prototype

```
U32 SSL_SetLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to enable.

Return value

The log filter mask before replacement.

Additional information

This function sets the log filter mask to use, entirely replacing the previously set filter mask. The bits set in the filter mask enable appropriate messages to the log.

Example

This example sets the log filter to report only keys and cryptographic data with all others disabled.

```
SSL_SetLogFilter(SSL_LOG_KEYS | SSL_LOG_CRYPTO);
```

See also

- *Logging flags* on page 60
- *SSL_AddLogFilter* on page 121
- *SSL_RemoveLogFilter* on page 124

4.7.7 SSL_SetWarnFilter()

Description

Set the active warning filter.

Prototype

```
U32 SSL_SetWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
`FilterMask`	Filters to enable.

Return value

The warning filter mask before replacement.

Additional information

This function sets the warning filter mask to use, entirely replacing the previously set filter mask. The bits set in the filter mask enable appropriate warnings.

Example

This example sets the warning filter to report only alerts and cryptography warnings with all others disabled.

```
SSL_SetWarnFilter(SSL_WARN_ALERT | SSL_WARN_CRYPTO);
```

See also

- *Warning flags* on page 61
- *`SSL_AddWarnFilter`* on page 122.
- *`SSL_RemoveWarnFilter`* on page 125.

4.8 Internal functions, variables and data structures

Internal functions of emSSL are not explained here as they are not required to use emSSL. The application should not rely on any of the internal elements, since they may be subject to change. Only the documented API functions are guaranteed to remain unchanged and compatible in future versions of emSSL.

4.9 API evolution

This section describes the changes made to the API for this version of emSSL.

Version 2.52

The following are the API additions in this version:

- Added `SSL_CLIENT_ConfigMutualAuth()`.
- Added `SSL_SERVER_ConfigMutualAuth()`.
- Added `SSL_SIGNATURE_SIGN_Add()`.
- Added `SSL_SIGNATURE_SIGN_RSA_API`.
- Added `SSL_SIGNATURE_SIGN_ECDSA_API`.
- Added `SSL_SUITE_RSA_WITH_CAMELLIA_128_GCM_SHA256`.
- Added `SSL_SUITE_RSA_WITH_CAMELLIA_256_GCM_SHA384`.
- Added `SSL_CURVE_secp192k1`.
- Added `SSL_CURVE_secp224k1`.
- Added `SSL_CURVE_secp256k1`.

Version 2.50

The following are the API changes in this version:

- Added Curve25519 elliptic curve.
- Added Chacha20-Poly1305 cipher suites and AEAD cipher.

The following are now documented:

- `SSL_GetVersionText()`.
- `SSL_GetCopyrightText()`.

Version 2.42

The following are the API changes in this version:

- Added Brainpool curves.

The following are now documented:

- `SSL_SUITE_GetCipherName()`.
- `SSL_SUITE_GetIANASuiteName()`.
- `SSL_SUITE_GetKeyExchangeName()`.
- `SSL_SUITE_GetMACAlgorithmName()`.
- `SSL_SUITE_GetPKAlgorithmName()`.
- `SSL_SUITE_QueryNull()`.
- `SSL_SUITE_QueryPKAlgorithm()`.

Version 2.40

The following are the API changes in this version:

- Removed the API `SSL_PRF_Add`.
- Added Camellia, ARIA, and SEED cipher suites.
- Added Camellia, ARIA, and SEED ciphers.

Version 2.30

The following are the API changes in this version:

- Removed the requirement to add PRFs. The API `SSL_PRF_Add` remains for now but has no effect: it is scheduled for removal in a future release.
- Added `SSL_ROOT_CERTIFICATE_Add()` to install root certificates.
- Added `SSL_SESSION_SetFlags()` to set per-session configuration flags.
- Added capability to request and respond with client certificates.

Chapter 5

Configuring emSSL

emSSL is configurable. It is designed for both high performance and low memory usage.

This chapter describes the available compile-time and runtime configuration options.

emSSL’s functionality (i.e. its feature set) is completely configurable using runtime calls to select the way that emSSL performs as a client and a server. However, there are choices to be made between different implementations of some computationally intensive algorithms. At the source level you can trade speed of execution for a more compact implementation to reduce code space — this configuration is made using preprocessor symbols defined in a particular manner when compiling emSSL from source code.

Please refer to the chapter *Best practice* on page 236 for details on best practices when configuring SSL and TLS systems.

5.1 Compile-time configuration

5.1.1 Application data fragmentation

Default

```
#define SSL_MAX_APP_DATA_FRAGMENT_LEN   1024
```

Override

To define a non-default value, define this symbol in `SSL_Conf.h`.

Description

Set this preprocessor symbol to define the maximum length of an application data packet. Send requests with more than this quantity of data will be transparently fragmented across multiple application data packets by emSSL.

The valid range is from 1 to 16384 inclusive. The advantage of setting this value to a small value is that less memory is required by emSSL when encapsulating the packet for transmission using the TLS protocol. The disadvantage of a small value is that there will be more transmission overhead required by the encapsulation and, therefore, transmission is less efficient in terms of bandwidth.

5.1.2 Server session cache size

Default

```
#define SSL_SESSION_CACHE_SIZE   5
```

Override

To define a non-default value, define this symbol in `SSL_Conf.h`.

Description

Set this preprocessor symbol to define the number of entries in the session cache. Setting this symbol to zero disables the session cache.

5.1.3 Supported ticket lengths

Default

```
#define SSL_MAX_SESSION_TICKET_LEN    256
```

Override

To define a non-default value, define this symbol in `SSL_Conf.h`.

Description

This preprocessor symbol defines the maximum session ticket length supported by TLS client connections. If this is set to zero, session tickets (as a client) will not be supported by emSSL.

The session ticket is sent by the server and, therefore, the client has no control over its length. You should consider the servers you will be connecting to, as a client, and determine an appropriate ticket size. Ticket sizes vary considerably in length, from a few hundred bytes up to two kilobytes.

5.2 Runtime configuration

Before a secure connection is established, emSSL must be configured with the cipher suites and supporting algorithms needed for a secure connection. `SSL_X_Config.c` is configured to match the requirements of most applications and can be taken as an example. You must configure:

- Cipher suites that emSSL will negotiate for secure connections. See *Adding cipher suites* on page 132.
- Bulk encryption ciphers, such as DES and AES. See *Adding ciphers* on page 135.
- Hash and MAC algorithms, such as SHA and MD5. See *Adding MACs* on page 136.
- Signature verification schemes required. See *Adding public key signature verifiers* on page 136.
- Message signing schemes required. See *Adding public key message signers* on page 136.
- Signature schemes offered. See *Adding signature algorithms* on page 137.
- Elliptic curves, if using elliptic curve cryptography. See *Adding elliptic curves* on page 138.
- TLS protocol versions that you wish to support. See *Adding TLS protocols* on page 138.

Each cipher suite that you add and protocol version that you support will require supporting components to be configured as well.

5.2.1 Cipher suites explained

A TLS cipher suite is composed of three parts:

- The key exchange method, for instance RSA or ECDHE-ECDSA
- The bulk / AEAD cipher, for instance 3DES-EDE-CBC or AES-256-GCM
- The message authentication scheme, for instance MD5 or SHA384

These three parts are put together in an identifier of the form:

TLS--*keyexchange*--WITH--*bulkcipher*--*authentication*

For instance, a TLS cipher that combines ECDHE--ECDSA key exchange with an AES--256--CBC bulk cipher and uses SHA--384 message authentication would be:

- TLS--ECDHE--ECDSA--WITH--AES--256--CBC--SHA384

This is a lot of information packed into a single identifier. emSSL identifies each suite following that naming convention but replaces “TLS” with “SSL”, and replaces hyphens with underscores such that the resulting identifier is acceptable to a C compiler. So, the above suite becomes:

- `SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`

In theory you create a cipher suite by picking any combination of key exchange, cipher, and MAC, but there are good reasons that only a handful of combinations are in popular use — those good reasons include security analysis of the cryptographic strengths of each of the components and appropriate matching between the parts.

For each of the cipher suite parts, you must add an appropriate implementation of public key methods (*Adding public key signature verifiers* on page 136), bulk ciphers (*Adding ciphers* on page 135), and message authentication (*Adding MACs* on page 136).

5.2.2 Adding cipher suites

emSSL supports a number of cipher suites and each supported cipher suite must be added to emSSL.

To install cipher suites into emSSL, call `SSL_SUITE_Add`, specifying the suite to add, in `SSL_X_Config()`:

```
void SSL_X_Config(void) {
  SSL_SUITE_Add(&SSL_SUITE_ECDHE_RSA_WITH_AES_256_GCM_SHA384);
}
```

The suites supported by this version of emSSL are specified using standard naming conventions described earlier:

ECDHE-ECDSA

- SSL_SUITE_ECDHE_ECDSA_WITH_NULL_SHA
- SSL_SUITE_ECDHE_ECDSA_WITH_RC4_128_SHA
- SSL_SUITE_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CCM
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CCM
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_128_CCM_8
- SSL_SUITE_ECDHE_ECDSA_WITH_AES_256_CCM_8
- SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- SSL_SUITE_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
- SSL_SUITE_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256

ECDHE-RSA

- SSL_SUITE_ECDHE_RSA_WITH_NULL_SHA
- SSL_SUITE_ECDHE_RSA_WITH_RC4_128_SHA
- SSL_SUITE_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_ECDHE_RSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_ECDHE_RSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- SSL_SUITE_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

ECDH-RSA

- SSL_SUITE_ECDH_RSA_WITH_NULL_SHA
- SSL_SUITE_ECDH_RSA_WITH_RC4_128_SHA
- SSL_SUITE_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_ECDH_RSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_ECDH_RSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_ECDH_RSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_ECDH_RSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_ECDH_RSA_WITH_AES_256_CBC_SHA384
- SSL_SUITE_ECDH_RSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_ECDH_RSA_WITH_ARIA_128_CBC_SHA256
- SSL_SUITE_ECDH_RSA_WITH_ARIA_128_GCM_SHA256
- SSL_SUITE_ECDH_RSA_WITH_ARIA_256_CBC_SHA384

- SSL_SUITE_ECDH_RSA_WITH_ARIA_256_GCM_SHA384
- SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
- SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256
- SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384
- SSL_SUITE_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384

ECDH-ECDSA

- SSL_SUITE_ECDH_ECDSA_WITH_NULL_SHA
- SSL_SUITE_ECDH_ECDSA_WITH_RC4_128_SHA
- SSL_SUITE_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_ECDH_ECDSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
- SSL_SUITE_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384
- SSL_SUITE_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384
- SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
- SSL_SUITE_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384

DHE-RSA

- SSL_SUITE_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_DHE_RSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_DHE_RSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_DHE_RSA_WITH_AES_256_CBC_SHA256
- SSL_SUITE_DHE_RSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_DHE_RSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_DHE_RSA_WITH_AES_128_CCM
- SSL_SUITE_DHE_RSA_WITH_AES_256_CCM
- SSL_SUITE_DHE_RSA_WITH_AES_128_CCM_8
- SSL_SUITE_DHE_RSA_WITH_AES_256_CCM_8
- SSL_SUITE_DHE_RSA_WITH_ARIA_128_CBC_SHA256
- SSL_SUITE_DHE_RSA_WITH_ARIA_256_CBC_SHA384
- SSL_SUITE_DHE_RSA_WITH_ARIA_128_GCM_SHA256
- SSL_SUITE_DHE_RSA_WITH_ARIA_256_GCM_SHA384
- SSL_SUITE_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
- SSL_SUITE_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
- SSL_SUITE_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
- SSL_SUITE_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
- SSL_SUITE_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256

RSA

- SSL_SUITE_RSA_WITH_RC4_128_MD5
- SSL_SUITE_RSA_WITH_RC4_128_SHA
- SSL_SUITE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_SUITE_RSA_WITH_AES_128_CBC_SHA
- SSL_SUITE_RSA_WITH_AES_256_CBC_SHA
- SSL_SUITE_RSA_WITH_AES_128_CBC_SHA256
- SSL_SUITE_RSA_WITH_AES_256_CBC_SHA256
- SSL_SUITE_RSA_WITH_AES_256_GCM_SHA384
- SSL_SUITE_RSA_WITH_AES_128_GCM_SHA256
- SSL_SUITE_RSA_WITH_AES_128_CCM
- SSL_SUITE_RSA_WITH_AES_256_CCM
- SSL_SUITE_RSA_WITH_AES_128_CCM_8

- `SSL_SUITE_RSA_WITH_AES_256_CCM_8`
- `SSL_SUITE_RSA_WITH_ARIA_128_CBC_SHA256`
- `SSL_SUITE_RSA_WITH_ARIA_256_CBC_SHA384`
- `SSL_SUITE_RSA_WITH_ARIA_128_GCM_SHA256`
- `SSL_SUITE_RSA_WITH_ARIA_256_GCM_SHA384`
- `SSL_SUITE_RSA_WITH_CAMELLIA_128_CBC_SHA`
- `SSL_SUITE_RSA_WITH_CAMELLIA_256_CBC_SHA`
- `SSL_SUITE_RSA_WITH_CAMELLIA_128_CBC_SHA256`
- `SSL_SUITE_RSA_WITH_CAMELLIA_256_CBC_SHA256`
- `SSL_SUITE_RSA_WITH_CAMELLIA_128_GCM_SHA256`
- `SSL_SUITE_RSA_WITH_CAMELLIA_256_GCM_SHA384`

5.2.2.1 Suite preference order

The order in which suites are added defines the default preference order of suites in the negotiation phase of a client connection.

5.2.3 Adding ciphers

You must add the required bulk cipher implementation using `SSL_Cipher_Add` when configuring cipher suites. The bulk cipher implementations are:

- `SSL_CIPHER_AES_128_CBC_API`
- `SSL_CIPHER_AES_256_CBC_API`
- `SSL_CIPHER_AES_128_GCM_API`
- `SSL_CIPHER_AES_256_GCM_API`
- `SSL_CIPHER_AES_128_CCM_API`
- `SSL_CIPHER_AES_256_CCM_API`
- `SSL_CIPHER_AES_128_CCM_8_API`
- `SSL_CIPHER_AES_256_CCM_8_API`
- `SSL_CIPHER_ARIA_128_CBC_API`
- `SSL_CIPHER_ARIA_256_CBC_API`
- `SSL_CIPHER_ARIA_128_GCM_API`
- `SSL_CIPHER_ARIA_256_GCM_API`
- `SSL_CIPHER_CAMELLIA_128_CBC_API`
- `SSL_CIPHER_CAMELLIA_256_CBC_API`
- `SSL_CIPHER_CAMELLIA_128_GCM_API`
- `SSL_CIPHER_CAMELLIA_256_GCM_API`
- `SSL_CIPHER_DES_CBC_API`
- `SSL_CIPHER_3DES_EDE_CBC_API`
- `SSL_CIPHER_RC4_128_API`
- `SSL_CIPHER_CHACHA20_POLY1305_API`

These cipher implementations are written entirely in software and do not offer any acceleration beyond configuring AES or DES during compilation (see *Ciphers* on page 148).

Although the implementation of ciphers in emSSL is highly efficient, emSSL is designed to be highly modular in order to take full advantage of both hardware acceleration and vendor-optimized cryptography libraries. Please refer to *Configuring cryptography* on page 140 for further details.

Example

The cipher suite TLS-ECDHE-ECDSA-WITH-AES-256-CBC-SHA384 uses AES-256-CBC as the bulk cipher. In order to support this cipher suite correctly you must add appropriate support for AES-256-CBC with:

```
SSL_CIPHER_Add(&SSL_CIPHER_AES_256_CBC_API);
```

5.2.4 Adding MACs

You must add required message authentication code implementations using `SSL_MAC_Add` when configuring cipher suites. The MAC implementations are:

- `SSL_MAC_MD5_API`
- `SSL_MAC_SHA_API`
- `SSL_MAC_SHA224_API`
- `SSL_MAC_SHA256_API`
- `SSL_MAC_SHA384_API`
- `SSL_MAC_SHA512_API`

These MAC implementations are written entirely in software and do not offer any acceleration beyond configuring MD5 and SHA during compilation.

Although the implementation of MACs in emSSL is highly efficient, emSSL is designed to be highly modular in order to take full advantage of both hardware acceleration and vendor-optimized cryptography libraries. Please refer to *Configuring cryptography* on page 140 for further details.

Example

The cipher suite TLS-ECDHE-ECDSA-WITH-AES-256-CBC-SHA384 uses SHA-384 as the MAC. In order to support this cipher suite correctly you must add appropriate support for SHA-384 with:

```
SSL_MAC_Add(&SSL_MAC_SHA384_API);
```

5.2.5 Adding public key signature verifiers

You must add the required public key signature verifiers (according to the cipher suites you select) using `SSL_SIGNATURE_VERIFY_Add()`. The public key signature verifier implementations are:

- `SSL_SIGNATURE_VERIFY_RSA_API`
- `SSL_SIGNATURE_VERIFY_DSA_API`
- `SSL_SIGNATURE_VERIFY_ECDSA_API`

These signature verification implementations are written entirely in software but take advantage of any hardware acceleration offered by the MAC schemes.

Although the implementation of signature verifiers in emSSL is highly efficient, emSSL is designed to be highly modular in order to take full advantage of both hardware acceleration and vendor-optimized cryptography libraries. Please refer to *Configuring cryptography* on page 140 for further details.

Example

The cipher suite TLS-ECDHE-ECDSA-WITH-AES-256-CBC-SHA384 uses the key exchange ECDHE-ECDSA where ECDSA is the signature method. In order to support this cipher suite correctly you must add appropriate support for ECDSA using:

```
SSL_SIGNATURE_VERIFY_Add(&SSL_SIGNATURE_VERIFY_ECDSA_API);
```

5.2.6 Adding public key message signers

You must add the required public key message signers verifiers (according to the cipher suites you select) using `SSL_SIGNATURE_SIGN_Add()`. The public key signature verifier implementations are:

- `SSL_SIGNATURE_VERIFY_RSA_API`
- `SSL_SIGNATURE_VERIFY_ECDSA_API`

These message signing implementations are written entirely in software but take advantage of any hardware acceleration offered by the MAC schemes.

Although the implementation of message signers in emSSL is highly efficient, emSSL is designed to be highly modular in order to take full advantage of both hardware acceleration and vendor-optimized cryptography libraries. Please refer to *Configuring cryptography* on page 140 for further details.

Example

The cipher suite TLS-RSA-WITH-AES-256-CBC-SHA384 uses a static RSA key exchange scheme and *in server mode* must sign part of the exchange between client and server. In order to support this cipher suite correctly you must add appropriate support for static RSA key exchange using:

```
SSL_SIGNATURE_VERIFY_Add(&SSL_SIGNATURE_SIGN_RSA_API);
```

5.2.7 Adding signature algorithms

When implementing emSSL as a client, the client must advertise the signature schemes that it is willing to accept when negotiating keys and verifying certificates. You must therefore configure the signature schemes that you wish to offer and also install the signature verifiers and MACs that comprise the signature algorithms you advertise.

The signature schemes are broken down by public key:

RSA

- SSL_SIGNATURE_MD5_WITH_RSA_ENCRYPTION
- SSL_SIGNATURE_SHA_WITH_RSA_ENCRYPTION
- SSL_SIGNATURE_SHA224_WITH_RSA_ENCRYPTION
- SSL_SIGNATURE_SHA256_WITH_RSA_ENCRYPTION
- SSL_SIGNATURE_SHA384_WITH_RSA_ENCRYPTION
- SSL_SIGNATURE_SHA512_WITH_RSA_ENCRYPTION

DSA

- SSL_SIGNATURE_SHA_WITH_DSA

ECDSA

- SSL_SIGNATURE_SHA_WITH_ECDSA
- SSL_SIGNATURE_SHA224_WITH_ECDSA
- SSL_SIGNATURE_SHA256_WITH_ECDSA
- SSL_SIGNATURE_SHA384_WITH_ECDSA
- SSL_SIGNATURE_SHA512_WITH_ECDSA

These signature algorithm implementations are written entirely in software and do not offer any acceleration beyond that offered by static configuration of each SHA and MD5 component.

Although the implementation of signature algorithms in emSSL is highly efficient, emSSL is designed to be highly modular in order to take full advantage of both hardware acceleration and vendor-optimized cryptography libraries. Please refer to *Configuring cryptography* on page 140 for further details.

Example

The cipher suite TLS-ECDHE-ECDSA-WITH-AES-256-CBC-SHA384 would use a signature based on ECDSA. You can add support for this scheme using a SHA-256 message digest and advertise it to a client with:

```
SSL_SIGNATURE_ALGORITHM_Add(SSL_SIGNATURE_SHA256_WITH_ECDSA);
```

However, you also need to add appropriate MAC algorithms and public key signature verifiers to support this scheme:

```
SSL_MAC_Add(&SSL_MAC_SHA256_API);
```

```
SSL_SIGNATURE_VERIFY_Add(&SSL_SIGNATURE_VERIFY_ECDSA_API);
```

5.2.8 Adding elliptic curves

If emSSL is configured to support any elliptic curve cipher suite, i.e. those starting ECDH or ECDHE, it must also be configured with at least one elliptic curve.

emSSL supports recommended NIST elliptic curves over prime fields which provide different encryption strengths.

The following NIST elliptic curves are implemented within emSSL, defined in the NIST standard FIPS 186-4:

- secp192r1 or P-192
- secp224r1 or P-224
- secp256r1 or P-256
- secp384r1 or P-384
- secp521r1 or P-521

The following NIST curves are implemented, but are significantly slower than the NIST curves above for the same security strength:

- secp192k1
- secp224k1
- secp256k1

The following Brainpool curves are implemented, but are significantly slower than the NIST curves for the same security strength:

- brainpoolP256r1
- brainpoolP384r1
- brainpoolP512r1

The following curve is implemented, although support for it is somewhat limited on the Internet:

- curve25519

Higher strength elliptic curves require more memory and more computational power to work with and many servers limit the elliptic curves they support in order to reduce the load on the server when establishing a secure connection. The client is burdened in a similar fashion, so you should carefully consider which curves to install to provide the required level of security and acceptable performance along with interoperability—see *Implementation details* on page 244 for some advice.

Example

To install the elliptic curves into emSSL, call `SSL_CURVE_Add()` in `SSL_X_Config()`:

```
void SSL_X_Config(void) {
  SSL_CURVE_Add(&SSL_CURVE_secp192r1);
  SSL_CURVE_Add(&SSL_CURVE_secp224r1);
  SSL_CURVE_Add(&SSL_CURVE_secp256r1);
  SSL_CURVE_Add(&SSL_CURVE_secp384r1);
  SSL_CURVE_Add(&SSL_CURVE_secp521r1);
}
```

5.2.9 Adding TLS protocols

You must configure the TLS protocol versions that you wish to support as a client or server. emSSL supports TLS protocol version 1.0, 1.1, and 1.2. You can add support for each protocol using `SSL_PROTOCOL_Add()`. The protocols offered by emSSL are:

- `SSL_PROTOCOL_TLS1v0_API`
- `SSL_PROTOCOL_TLS1v1_API`
- `SSL_PROTOCOL_TLS1v2_API`

Note that adding support for a particular version of the protocol has an effect on the TLS pseudorandom function that is required.

TLS version 1.0 and 1.1

These schemes require the TLS version 1.0 PRF implementation that is a combination of MD5 and SHA.

TLS version 1.2

This protocol version will always use PRF-SHA256 as the PRF for all TLS 1.0 and 1.1 cipher suites. For TLS 1.2 cipher suites, the PRF is defined by the cipher suite itself:

- If the cipher suite ends with -SHA256, PRF-SHA256 is required
- If the cipher suite ends with -SHA384, PRF-SHA384 is required

Example

We support a single cipher suite SSL-SUITE-RSA-WITH-AES-128-CBC-SHA and TLS protocol version 1.0 only. Putting aside configuration of the cipher suite and other items and concentrating on the PRF, this cipher suite and TLS protocol combination requires only the PRF-TLS1 pseudorandom function:

```
SSL_SUITE_Add(&SSL_SUITE_RSA_WITH_AES_128_CBC_SHA);
SSL_PROTOCOL_ADD(&SSL_PROTOCOL_TLS1v0_API);
```

Extending support for TLS versions 1.1 and 1.2 but retaining a single TLS version 1.0 cipher suite requires the addition of PRF-SHA256 — this is because the TLS version 1.0 cipher suite's MAC is upgraded to SHA-256 when negotiating a TLS 1.2 connection:

```
SSL_SUITE_Add(&SSL_SUITE_RSA_WITH_AES_128_CBC_SHA);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v0_API);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v1_API);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v2_API);
```

Extending support for the TLS 1.2 cipher suite SSL-SUITE-RSA-WITH-AES-256-GCM-SHA384 requires the addition of PRF-SHA384 as this is mandated by the cipher suite's SHA-384 MAC:

```
SSL_SUITE_Add(&SSL_SUITE_RSA_WITH_AES_128_CBC_SHA);
SSL_SUITE_Add(&SSL_SUITE_RSA_WITH_AES_256_GCM_SHA384);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v0_API);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v1_API);
SSL_PROTOCOL_Add(&SSL_PROTOCOL_TLS1v2_API);
```

The order that you add components to emSSL does not matter, you can add components in the order that feels most natural. When `SSL_X_Config()` returns, `SSL_Init()` will validate the configuration and warn you on anything that can be eliminated and will halt if it requires something that has not been configured.

emSSL will diagnose issues if installed TLS support requires an underlying algorithm that is not configured at the emCrypt layer.

Chapter 6

Configuring cryptography

6.1 Overview

In addition to configuring emSSL capabilities at the protocol level, it is necessary to configure how these are implemented by the shared cryptographic library, emCrypt.

emCrypt provides cryptographic services for all SEGGER security products (emSSL, emSSH, emSecure-RSA, and emSecure-ECDSA) and must be configured before it is used standalone or with one of these products.

There are two parts to emCrypt configuration:

- *Compile-time configuration* which defines the static configuration of the software, selecting the way each particular algorithm is implemented. Particular configurations are selected by setting various preprocessor symbols when compiling.
- *Runtime configuration* where the cryptographic algorithms are configured for higher-level clients such as emSSL and emSSH. Particilar configurations are selected by installing cryptographic services when initializing the cryptographic library.

The following sections describe compile-time and runtime configuation of emCrypt.

6.2 Compile-time configuration

In order to configure how compute-intensive cryptographic algorithms are compiled to balance code size and execution performance you can change the compile-time flags in the configuration file `CRYPTO_Conf.h`.

The default configuration strikes a balance between code size and execution speed and runs well on the broadest range of hardware.

Note

All user configuration of the cryptographic algorithms must be made in the file `CRYPTO_Conf.h` *and only that file*. Do not make any adjustments to the file `CRYPTO_ConfDefaults.h`.

6.2.1 Multiprecision integer, bits per limb

Default

```
#define CRYPTO_MPI_BITS_PER_LIMB     32
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

This preprocessor symbol configures the number of bits per limb for multiprecision integer algorithms. The default of 32 matches 32-bit targets well, such as ARM and PIC32. In general, it is best to set the number of bits per limb to the number of bits in the standard `int` or `unsigned` type used by the target compiler.

Supported configurations are:

- 32 — requires the target compiler to support 64-bit types natively (i.e. `unsigned long long` or `unsigned __int64`),
- 16 — which should run on any ISO compiler whose native integer types are 16 or 32 bit and supports 32-bit `unsigned long`.
- 8 — 8-bit limb sizes are supported and selecting this size may well lead to better multiplication performance on 8-bit architectures.

6.2.2 Hashes

6.2.2.1 MD5

Default

```
#define CRYPTO_CONFIG_MD5_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the MD5 hash functions for size rather than for speed. When optimized for speed, the MD5 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	Flash	0.3 KB	0.4 KB	0.7 KB
1	0.16 KB	-	-	2.0 KB	2.0 KB

6.2.2.2 SHA-1

Default

```
#define CRYPTO_CONFIG_SHA1_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-1 hash functions for size rather than for speed. When optimized for speed, the SHA-1 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M4 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	-	-	0.6 KB	0.6 KB
1	0.16 KB	-	-	3.6 KB	3.6 KB

6.2.2.3 SHA-256

Default

```
#define CRYPTO_CONFIG_SHA256_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-256 hash functions for size rather than for speed. When optimized for speed, the SHA-256 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.17 KB	Flash	0.3 KB	0.5 KB	0.8 KB
1	0.17 KB	-	-	7.7 KB	7.7 KB

6.2.2.4 SHA-512

Default

```
#define CRYPTO_CONFIG_SHA512_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-512 hash functions for size rather than for speed. When optimized for speed, the SHA-512 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.20 KB	Flash	0.7 KB	1.1 KB	1.8 KB
1	0.20 KB	Flash	0.7 KB	10.3 KB	11.0 KB
2	0.20 KB	Flash	0.1 KB	41.5 KB	41.6 KB

6.2.3 Ciphers

6.2.3.1 AES

Default

```
#define CRYPTO_CONFIG_AES_OPTIMIZE     2
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize AES to use tables for matrix multiplication. Optimization levels are 0 through 7 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.24 KB	Flash	2.0 KB	3.2 KB	5.2 KB
1	0.24 KB	Flash	2.0 KB	2.7 KB	4.7 KB
2	0.24 KB	Flash	8.5 KB	2.4 KB	10.9 KB
3	0.24 KB	Flash	1.9 KB	12.5 KB	14.4 KB
4	0.24 KB	RAM	2.0 KB	3.2 KB	5.2 KB
5	0.24 KB	RAM	2.0 KB	2.7 KB	4.7 KB
6	0.24 KB	RAM	8.5 KB	2.4 KB	10.9 KB
7	0.24 KB	RAM	1.9 KB	12.5 KB	14.4 KB

6.2.3.2 DES

Default

```
#define CRYPTO_CONFIG_DES_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize DES and 3DES to place tables in RAM rather than flash. Optimization levels are 0 through 5 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.38 KB	Flash	2.1 KB	1.3 KB	3.4 KB
1	0.38 KB	Flash	2.1 KB	2.1 KB	4.2 KB
2	0.38 KB	Flash	2.1 KB	5.3 KB	7.4 KB
3	0.38 KB	RAM	2.1 KB	1.3 KB	3.4 KB
4	0.38 KB	RAM	2.1 KB	2.1 KB	4.2 KB
5	0.38 KB	RAM	2.1 KB	5.3 KB	7.4 KB

6.2.3.3 SEED

Default

```
#define CRYPTO_CONFIG_SEED_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize SEED to place tables in RAM rather than flash and to optimized the table sizes. Optimization levels are 0 through 3 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.14 KB	Flash	0.5 KB	0.5 KB	1.0 KB
1	0.14 KB	Flash	4.0 KB	0.4 KB	4.4 KB
2	0.14 KB	RAM	0.5 KB	0.5 KB	1.0 KB
3	0.14 KB	RAM	4.0 KB	0.4 KB	4.4 KB

6.2.3.4 ARIA

Default

```
#define CRYPTO_CONFIG_ARIA_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize ARIA to place tables in RAM rather than flash.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.28 KB	Flash	1.0 KB	1.9 KB	2.9 KB
1	0.28 KB	RAM	1.0 KB	1.9 KB	2.9 KB

6.2.3.5 Camellia

Default

```
#define CRYPTO_CONFIG_CAMELLIA_OPTIMIZE     0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Camellia to use more efficient tables. Optimization levels are 0 (smallest) to 3 (fastest).

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.27 KB	Flash	1.0 KB	28.8 KB	29.8 KB
1	0.27 KB	Flash	4.0 KB	20.7 KB	24.7 KB
2	0.27 KB	RAM	1.0 KB	28.8 KB	29.8 KB
3	0.27 KB	RAM	4.0 KB	20.7 KB	24.7 KB

6.3 Runtime configuration

The ciphers and hash functions that clients require must be installed as part of emCrypt configuration. The function `CRYPTO_X_Conf()` is called from `CRYPTO_Init()` to install any required cryptographic support.

emCrypt provides software implementations of all ciphers and hashes, but also supports plug-in hardware accelerators.

6.3.1 Hashes

emCrypt provides the following software implementations of the following hash algorithms for emSSL:

- MD5
- SHA-1
- SHA-256
- SHA-512

This section summarizes how to install the software hash implementations. For details on how to plug in hardware-assisted hash algorithms for a particular device, see *Plug-in hardware accelerators* on page 162.

6.3.1.1 MD5

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_MD5_SW;
```

Description

This API provides a software-only implementation of MD5.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_MD5_Install(&CRYPTO_HASH_MD5_SW, NULL);
}
```

See also

See *MD5* on page 144 for details on how to configure the performance and footprint of this algorithm.

6.3.1.2 SHA-1

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_SW;
```

Description

This API provides a software-only implementation of SHA-1.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_SW, NULL);
}
```

See also

See *SHA-1* on page 145 for details on how to configure the performance and footprint of this algorithm.

6.3.1.3 SHA-256

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_SW;
```

Description

This API provides a software-only implementation of SHA-256 and SHA-224.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_SW, NULL);
}
```

See also

See *SHA-256* on page 146 for details on how to configure the performance and footprint of this algorithm.

6.3.1.4 SHA-512

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA512_SW;
```

Description

This API provides a software-only implementation of SHA-512 and SHA-384.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA512_Install(&CRYPTO_HASH_SHA512_SW, NULL);
}
```

See also

See *SHA-512* on page 147 for details on how to configure the performance and footprint of this algorithm.

6.3.2 Ciphers

emCrypt provides the following software implementations of the following cipher algorithms for emSSL:

- AES
- DES
- SEED
- ARIA
- Camellia

This section summarizes how to install the software cipher implementations. For details on how to plug in hardware-assisted cipher algorithms for a particular device, see *Plug-in hardware accelerators* on page 162.

6.3.2.1 AES

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_SW;
```

Description

This API provides a software-only implementation of AES.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_SW, NULL);
}
```

See also

See *AES* on page 148 for details on how to configure the performance and footprint of this algorithm.

6.3.2.2 DES

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_SW;
```

Description

This API provides a software-only implementation of DES.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_TDES_Install(&CRYPTO_CIPHER_TDES_SW, NULL);
}
```

See also

See *DES* on page 149 for details on how to configure the performance and footprint of this algorithm.

6.3.2.3 SEED

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_SEED_SW;
```

Description

This API provides a software-only implementation of SEED.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SEED_Install(&CRYPTO_CIPHER_SEED_SW, NULL);
}
```

See also

See *SEED* on page 150 for details on how to configure the performance and footprint of this algorithm.

6.3.2.4 ARIA

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_ARIA_SW;
```

Description

This API provides a software-only implementation of ARIA.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_ARIA_Install(&CRYPTO_CIPHER_ARIA_SW, NULL);
}
```

See also

See *ARIA* on page 151 for details on how to configure the performance and footprint of this algorithm.

6.3.2.5 Camellia

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_CAMELLIA_SW;
```

Description

This API provides a software-only implementation of Camellia.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_CAMELLIA_Install(&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
}
```

See also

See *Camellia* on page 152 for details on how to configure the performance and footprint of this algorithm.

6.3.3 Plug-in hardware accelerators

SEGGER security products are written in a way such that underlying cryptographic operations can be exchanged in order to benefit from hardware acceleration or vendor libraries optimized for a particular device.

emSSL requires no additional hardware in order to execute its underlying cryptographic operations: public key algorithms, bulk encipherment, and message authentication are completely implemented in software. However, there are many devices that offer hardware acceleration for one or more of these operations and there is nothing that prevents emSSL from utilizing any such capability.

For further information on hardware acceleration, refer to the following sections:

- *LPC18S and LPC43S AES ROM (Add-on)* on page 163
- *Kinetis CAU coprocessor (Add-on)* on page 165
- *STM32 CRYP coprocessor (Add-on)* on page 171
- *STM32 AES coprocessor (Add-on)* on page 174
- *STM32 HASH coprocessor (Add-on)* on page 175
- *EFM32 CRYPTO coprocessor (Add-on)* on page 178

For background information on hardware acceleration, refer to *Hardware acceleration* on page 251.

6.3.3.1 LPC18S and LPC43S AES ROM (Add-on)

The LPC18S*xx* and LPC43S*xx* microcontrollers provide an AES-128 hardware accelerator. The capabilities of this accelerator are exposed through a ROM-based API which insulates the programmer from changes to or variants of the underlying accelerator hardware.

emSSL has specialized hardware-assisted AES ciphering for the following cryptographic algorithms:

- AES-128 in ECB and CBC modes.

All other AES-128 cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software managing the cipher mode. All ciphering with AES-192 and AES-256 falls back to using a pure software AES kernel.

6.3.3.1.1 Installing LPC ROM hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_LPC_ROM;
```

If all you require is AES-128, you can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM, 0);
}
```

However, if you require AES-192 or AES-256 in addition to AES-128, you must install a software fallback for these key sizes:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM,
                     &CRYPTO_CIPHER_AES_SW);
}
```

6.3.3.1.2 LPC cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.1.3 Sample LPS18S setup

The following is the cryptographic setup for the NXP LPCXpresso18S37 board:

```
/*********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_LPC18S37.c
Purpose     : Configure CRYPTO for LPC18S37 devices.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
 CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES_HW_LPC_ROM, &CRYPTO_CIPHER_AES_SW);
 CRYPTO_TDES_Install      (&CRYPTO_CIPHER_TDES_SW,        0);
 CRYPTO_MD5_Install       (&CRYPTO_HASH_MD5_SW,           0);
 CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_SW,          0);
 CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,        0);
 CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,        0);
 CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,     0);
}

/*************************** End of file ****************************/
```

6.3.3.2 Kinetis CAU coprocessor (Add-on)

The Kinetis Cryptographic Acceleration Unit (CAU) is a primitive accelerator presented as a memory-mapped peripheral.

emSSL has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the CAU:

- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.
- MD5
- SHA-1
- SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software manging the cipher mode.

6.3.3.2.1 Installing CAU hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_Kinetis_CAU;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_MD5_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA1_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA224_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_Kinetis_CAU;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_MD5_Install   (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,    NULL);
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,   NULL);
  CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_Kinetis_CAU, NULL);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_Kinetis_CAU, NULL);
  CRYPTO_AES_Install   (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,  NULL);
  CRYPTO_TDES_Install  (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, NULL);
}
```

> **Note**
>
> Whilst there is an MD5 accelerator, hardware-assisted MD5 is slower than a pure software implementation of MD5 using Thumb-2 so we recommend that you do not install the MD5 accelerator.

6.3.3.2.2 Kinetis cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.2.3 Sample Kinetis setup

The following is the cryptographic setup for the SEGGER emPower board based on the Kinetis K66 device.

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------
File        : CRYPTO_X_Config_K66.c
Purpose     : Configure CRYPTO for K66 devices.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Install hardware assistance.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,    NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,   NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_HW_Kinetis_CAU, NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_HW_Kinetis_CAU, NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, NULL);
  //
  // Software ciphers.
  //
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Software hashing.
  //
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,    NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
  //
  // Turn on clocks to RNGA, bit 0 of SIM_SCGC3, and install RNG.
  //
  pReg = (void *)0x40048030;
  *pReg |= 1;
  //
  // Install Hash_DRBG-SHA-256 with RNGA entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_Kinetis_RNGA);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.3.3.3 iMX RT10xx data coprocessor (Add-on)

The iMX RT10xx Data Coprocessor (DCP) is a programmable crotographic accelerator presented as a memory-mapped peripheral.

emSSL has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the DCP:

- AES-128 in ECB and CBC modes.
- SHA-1
- SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software manging the cipher mode.

6.3.3.3.1 Installing iMX RT10xx hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA1_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_RT10xx_DCP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,
                        &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_RT10xx_DCP,
                        &CRYPTO_HASH_SHA256_SW);
  CRYPTO_AES_Install   (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,
                        &CRYPTO_CIPHER_AES_SW);
  //
  // Install Hash_DRBG-SHA-256 with TRNG entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                       &CRYPTO_RNG_HW_RT10xx_TRNG);
}
```

6.3.3.3.2 RT10xx cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.3.3 Sample Kinetis setup

The following is the cryptographic setup for the SEGGER RT1051 Trace Reference board.

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_RT10xx.c
Purpose     : Configure CRYPTO for iMX RT10xx devices.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  //
  // Install hardware assistance.
  //
//CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,
 &CRYPTO_HASH_SHA1_SW);    NXP investigating issue with DCP
//CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_HW_RT10xx_DCP,
 &CRYPTO_HASH_SHA256_SW);  NXP investigating issue with DCP
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,
  &CRYPTO_CIPHER_AES_SW);
  //
  // Software ciphers.
  //
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Software hashing.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,       NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,      NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,    NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,    NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,    NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
  //
  // Install Hash_DRBG-SHA-256 with TRNG entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_RT10xx_TRNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.3.3.4 STM32 CRYP coprocessor (Add-on)

The STM32 cryptographic processor (CRYP) is a capable hardware accelerator presented as a memory-mapped peripheral that accelerates AES and TDES encryption and decryption. There are two variants of the CRYP processor with different capabilities present on the following family members:

- STM32F41x CRYP, hereafter referred to as the *standard CRYP processor*, and
- STM32F43x/F47x CRYP, hereafter referred to as the *enhanced CRYP processor*.

emSSL has support for the following cryptographic algorithms using both CRYP variants:

- DES in ECB and CBC modes.
- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.

For the enhanced CRYP processor, direct acceleration is provided for:

- AES-128, AES-192, and AES-256 in CCM(12,4) and GCM(12,4) modes.

For the standard CRYP processor, acceleration is provided for:

- AES-128, AES-192, and AES-256 ciphering with GCM and CCM in software.

For CCM and GCM modes, the CRYP processor supports only fixed 16-byte authentication tags and 12-byte IVs with 4-byte counters. Therefore, AES-CCM acceleration is not immediately suitable for authenticated encryption in SSH as SSH requires zero-length IVs with 16-byte counters.

6.3.3.4.1 Installing CRYP hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_CRYP;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_STM32_CRYP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_CRYP);
  CRYPTO_TDES_Install(&CRYPTO_CIPHER_TDES_HW_STM32_CRYP);
}
```

6.3.3.4.2 Enabling the CRYP coprocessor

You must enable clocks and reset the CRYP peripheral before reading or writing its registers. For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
*pReg |= 1U << 4;                   // RCC_AHB2ENR.CRYPEN=1
pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
*pReg |= 1U << 4;                   // RCC_AHB2RSTR.CRYPRST=1
*pReg &= ~(1U << 4);                // RCC_AHB2RSTR.CRYPRST=0
```

6.3.3.4.3 STM32 cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.4.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```
/*********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG             *
*                         The Embedded Experts                        *
*                            www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_STM32F75x.c
Purpose     : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Turn on clocks to the CRYP accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 4;                   // RCC_AHB2ENR.CRYPEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 4;                   // RCC_AHB2RSTR.CRYPRST=1
  *pReg &= ~(1u << 4);                // RCC_AHB2RSTR.CRYPRST=0
  //
  // Install cipher hardware assistance.
  //
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
  //
  // Turn on clocks to the HASH accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 5;                   // RCC_AHB2ENR.HASHEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 5;                   // RCC_AHB2RSTR.HASHRST=1
  *pReg &= ~(1u << 5);                // RCC_AHB2RSTR.HASHRST=0
  //
  // Install hardware hashing with software fallback (required).
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
  &CRYPTO_HASH_MD5_SW);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
  &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA224_Install
  (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
  //
  // Software hashing.
  //
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Turn on clocks to the RNG and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 6;                   // RCC_AHB2ENR.RNGEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 6;                   // RCC_AHB2RSTR.RNGRST=1
  *pReg &= ~(1u << 6);                // RCC_AHB2RSTR.RNGRST=0
  //
  // Random number generator.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG, &CRYPTO_RNG_HW_STM32_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.3.3.5 STM32 AES coprocessor (Add-on)

The STM32 AES hardware accelerator (AES) is a hardware accelerator presented as a memory-mapped peripheral that accelerates AES-128 and AES-256 encryption and decryption. The AES accelerator is present on selected STM32L4 devices.

emSSL has support for the following cryptographic algorithms using the AES hardware accelerator:

- AES-128 and AES-256 in ECB and CBC modes.

6.3.3.5.1 Installing AES hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_AES;
```

You can install hardware support for AES-128 and AES-192 *only* using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES, NULL);
}
```

If you require AES-192 support, you must install a software fallback that is used when ciphering with a 192-bit key:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES,
                      &CRYPTO_CIPHER_AES_SW);
}
```

6.3.3.5.2 Enabling the AES coprocessor

You must enable clocks and reset the AES peripheral before reading or writing its registers. For the STM32L4A6 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x4002104C;  // RCC_AHB2ENR
*pReg |= 1U << 4;                   // RCC_AHB2ENR.AESEN=1
pReg = (volatile U32 *)0x4002102C;  // RCC_AHB2RSTR
*pReg |= 1U << 16;                  // RCC_AHB2RSTR.AESRST=1
*pReg &= ~(1U << 16);               // RCC_AHB2RSTR.AESRST=0
```

6.3.3.5.3 STM32 cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.6 STM32 HASH coprocessor (Add-on)

The STM32 hash coprocessor (HASH) is a hardware accelerator presented as a memory-mapped peripheral that accelerates calculation of MD5, SHA-1, SHA-224 and SHA-256 message digests.

emSSL has HASh accelerator support for the following cryptographic algorithms:

- MD5 message digest.
- SHA-1 message digest.
- SHA-224 and SHA-256 message digest.

6.3.3.6.1 Installing HASH hardware support

The following interfaces are provided:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_MD5_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA224_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_HW_STM32_HASH;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_MD5_Install   (&CRYPTO_HASH_MD5_HW_STM32_HASH);
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_STM32_HASH);
  CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_STM32_HASH);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_STM32_HASH);
}
```

6.3.3.6.2 Enabling the HASH coprocessor

You must enable clocks and reset the HASH peripheral before reading or writing its registers.

For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
*pReg |= 1u << 5;                   // RCC_AHB2ENR.HASHEN=1
pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
*pReg |= 1u << 5;                   // RCC_AHB2RSTR.HASHRST=1
*pReg &= ~(1u << 5);                // RCC_AHB2RSTR.HASHRST=0
```

For the STM32L4 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *RCC_AHB2RSTR = (U32 *)0x4002102C;
volatile U32 *RCC_AHB2ENR  = (U32 *)0x4002104C;
//
*RCC_AHB2ENR  |= 1<<17;
*RCC_AHB2RSTR |= 1<<17;
*RCC_AHB2RSTR &= ~(1<<17);
```

6.3.3.6.3 STM32 cryptographic units

The emSSL implementation of hardware assistance requires two cryptographic units with indexes #0 and #1 that cover ciphering (unit #0) and hashing (unit #1). See *CRYPTO-OS integration* on page 222 for further details.

6.3.3.6.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```
/*********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                         The Embedded Experts                       *
*                            www.segger.com                          *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_STM32F75x.c
Purpose     : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Turn on clocks to the CRYP accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 4;                   // RCC_AHB2ENR.CRYPEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 4;                   // RCC_AHB2RSTR.CRYPRST=1
  *pReg &= ~(1u << 4);                // RCC_AHB2RSTR.CRYPRST=0
  //
  // Install cipher hardware assistance.
  //
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
  //
  // Turn on clocks to the HASH accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 5;                   // RCC_AHB2ENR.HASHEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 5;                   // RCC_AHB2RSTR.HASHRST=1
  *pReg &= ~(1u << 5);                // RCC_AHB2RSTR.HASHRST=0
  //
  // Install hardware hashing with software fallback (required).
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
  &CRYPTO_HASH_MD5_SW);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
  &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA224_Install
  (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
  //
  // Software hashing.
  //
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Turn on clocks to the RNG and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 6;                   // RCC_AHB2ENR.RNGEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 6;                   // RCC_AHB2RSTR.RNGRST=1
  *pReg &= ~(1u << 6);                // RCC_AHB2RSTR.RNGRST=0
  //
  // Random number generator.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG, &CRYPTO_RNG_HW_STM32_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.3.3.7 EFM32 CRYPTO coprocessor (Add-on)

The EFM32 cryptographic coprocessor (CRYPTO) is presented as a memory-mapped peripheral.

emSSL has specialized hardware-assisted hashing support for the following cryptographic algorithms using the CRYPTO coprocessor:

- SHA-1

6.3.3.7.1 Installing CRYPTO hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, NULL);
}
```

6.3.3.7.2 EFM32 cryptographic units

The emSSL implementation of hardware assistance requires one cryptographic unit with index #0 covering hashing and RSA operations. See *CRYPTO-OS integration* on page 222 for further details.

If you wish to reduce power consumption, it is possible to enable and clocks to the crypto unit when `CRYPTO_OS_Claim()` is called and disable them `CRYPTO_OS_Unclaim()` is called (for cryptographic unit #0).

6.3.3.7.3 Modular exponentiation API

Function	Description
Windowing, Montgomery reduction	
`CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 2-bit window.
`CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 3-bit window.
`CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 4-bit window.
`CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 5-bit window.
`CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 6-bit window.
`CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 2-bit window.
`CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 3-bit window.
`CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 4-bit window.
`CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 5-bit window.
`CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO()`	Modular exponentiation, Montgomery reduction, 6-bit window.

6.3.3.7.3.1 CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO
                                                  (      CRYPTO_MPI         * pSelf,
                                                   const CRYPTO_MPI         * pExponent,
                                                   const CRYPTO_MPI         * pModulus,
                                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.2 CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO
                                             (      CRYPTO_MPI         * pSelf,
                                              const CRYPTO_MPI         * pExponent,
                                              const CRYPTO_MPI         * pModulus,
                                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.3 CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO
                                                      (      CRYPTO_MPI         * pSelf,
                                                       const CRYPTO_MPI         * pExponent,
                                                       const CRYPTO_MPI         * pModulus,
                                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.4 CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO
                                                 (      CRYPTO_MPI         * pSelf,
                                                  const CRYPTO_MPI         * pExponent,
                                                  const CRYPTO_MPI         * pModulus,
                                                        CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.5 CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO
                                                    (      CRYPTO_MPI         * pSelf,
                                                     const CRYPTO_MPI         * pExponent,
                                                     const CRYPTO_MPI         * pModulus,
                                                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.6 CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO
                                                  (      CRYPTO_MPI         * pSelf,
                                                   const CRYPTO_MPI         * pExponent,
                                                   const CRYPTO_MPI         * pModulus,
                                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.7 CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO
                                                    (      CRYPTO_MPI         * pSelf,
                                                     const CRYPTO_MPI         * pExponent,
                                                     const CRYPTO_MPI         * pModulus,
                                                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.8 CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO
                                              (      CRYPTO_MPI         * pSelf,
                                               const CRYPTO_MPI         * pExponent,
                                               const CRYPTO_MPI         * pModulus,
                                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.9 CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO
                                                  (      CRYPTO_MPI         * pSelf,
                                                   const CRYPTO_MPI         * pExponent,
                                                   const CRYPTO_MPI         * pModulus,
                                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
`pSelf`	Pointer to MPI that contains the base; exponential on return.
`pExponent`	Pointer to MPI that contains the exponent.
`pModulus`	Pointer to MPI that contains the modulus.
`pMem`	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.3.10 CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO
                                              (      CRYPTO_MPI         * pSelf,
                                               const CRYPTO_MPI         * pExponent,
                                               const CRYPTO_MPI         * pModulus,
                                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

6.3.3.7.4 Performance

6.3.3.7.4.1 SHA-1

Output from the benchmark `CRYPTO_Bench_SHA1` is shown below.

```
(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
SHA-1 Benchmark V2.00 compiled May 24 2017 12:06:22

Compiler: clang 4.0.0 (tags/RELEASE_400/final)
System:   Processor speed               = 19.000 MHz
Config:   CRYPTO_CONFIG_SHA1_OPTIMIZE    = 1
Config:   CRYPTO_CONFIG_SHA1_HW_OPTIMIZE = 1

+--------------+-----------+
| Algorithm    | Hash MB/s |
+--------------+-----------+
| SHA-1        |      0.76 |
| SHA-1 (HW)   |      6.77 |
+--------------+-----------+

Benchmark complete
```

6.3.3.7.5 Sample EFM32 setup

The following is the cryptographic setup for the Silicon Labs Pearl and Jade Gecko devices:

```
/*********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG             *
*                        The Embedded Experts                         *
*                           www.segger.com                            *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_EFM32.c
Purpose     : Configure CRYPTO for EFM32 Pearl and Jade Geckos.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *HFBUSCLKEN0;
  //
  CRYPTO_MD5_Install       (&CRYPTO_HASH_MD5_SW,               NULL);
  CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, NULL);
  CRYPTO_SHA224_Install    (&CRYPTO_HASH_SHA224_SW,            NULL);
  CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,            NULL);
  CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES,                NULL);
  CRYPTO_TDES_Install      (&CRYPTO_CIPHER_TDES,               NULL);
  CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,            NULL);
  CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,         NULL);
  //
  // Clock CRYPTO peripheral.
  //
  HFBUSCLKEN0 = (void *)0x400E40B0UL;
  *HFBUSCLKEN0 |= 1UL << 1;  // Turn on clock to CRYPTO unit
}

/*************************** End of file ****************************/
```

6.3.4 Secure random numbers

In order to guarantee the privacy of communications, it is vital that emSSL can call upon a stream of random numbers. Many microcontrollers that have Ethernet peripherals also provide cryptographic accelerators and true random number generators (RNGs), but not all do.

The sample implementation shipped for Windows use Microsoft's cryptographically-secure random number API to gather randomness, which satisfies emSSL's requirements.

For embedded targets, it isn't necessary to provide a fast random number generator, the hardware RNG will fit perfectly — connection time is dominated by public key operations, not tens of bytes of (relatively) slowly-gathered random data.

For devices that have no true random source, it suffices to gather a few hundred bits of random data from jitter in some physical timer or readings of the low order bits of some ADC, and feed that to a software random bit generator.

6.3.4.1 Installing random sources

The function `CRYPTO_RNG_Install` installs a source of randomness that the emCrypt component can use. The source of randomness can be either hardware or software, and the quality of that randomness is important.

6.3.4.1.1 Hardware-only random sources

emCrypt has add-on drivers for the Kinetis RNGA and STM32 RNG peripherals. You can install the hardware source as both the random bit generator and the source of entropy, for example:

```
CRYPTO_RNG_Install(&CRYPTO_RNG_HW_Kinetis_RNGA);
```

This only provides secure random data if the hardware produces secure random data: the STM32 RNG and Kinetis RNGA have provisos that the random data they produce are potentially not secure.

6.3.4.1.2 Secure random bit generator with hardware entropy

emCrypt supports using hardware sources of entropy to seed a deterministic random bit generator, and emCrypt fully implements the NIST DRBG random bit generators.

`CRYPTO_RNG_InstallEx` installs both a source of entropy and the random bit generator that it seeds: emCrypt will use the random bit generator to acquire random data and the entropy source feeds the random bit generator.

The DRBGs implemented are:

```
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA1;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA384;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512_224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512_256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA1;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA384;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512_224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512_256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_TDES;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES128;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES192;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES256;
```

The following installs both the DRBG and a source of entropy:

```
CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                     &CRYPTO_RNG_HW_Kinetis_RNGA);
```

6.4 Example configurations

6.4.1 Minimal Cortex-M configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Cortex-M devices:

```
/*********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                         The Embedded Experts                       *
*                            www.segger.com                          *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------
File        : CRYPTO_X_Config_SSL_CM.c
Purpose     : Configure CRYPTO for full SSL with no hardware
              accelerators and a dummy, insecure, random number
              generator.

Additional information:
  The dummy random number generator does not generate secure random
  numbers, but can be run on any hardware with memory at 0x20000000.
  To provide secure random numbers modify it according to the hardware
  capabilities.

  Random number generators for different hardware is available from
  SEGGER upon request.
*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Local functions
*
**********************************************************************
*/

/*********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get random data from RNG.
*
*  Parameters
*    pData   - Pointer to the object that receives the random data.
*    DataLen - Octet length of the random data.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  if (pData && DataLen) {
    while (DataLen--) {
      *pData++ = *((volatile U8*)0x20000000 + DataLen);
    }
  }
}

/*********************************************************************
*
```

```
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure no hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  //
  static const CRYPTO_RNG_API _RNG = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,     NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_SW,      NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 with "random" data from
  // RAM.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.4.2 Windows configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Windows:

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_Full_Win32.c
Purpose     : Configure full cryptography for x86 Win32.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#define _CRT_RAND_S  /*emDoc ignore*/
#include <stdlib.h>
#include <stdio.h>
#include "CRYPTO.h"

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get entropy from Win32 secure random API.
*
*  Parameters
*    pData   - Pointer to object that receives the random bitstream.
*    DataLen - Octet length of the object.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  unsigned V;
  unsigned L;
  //
  // Use Windows cryptographically-secure random number source.
  //
  while (DataLen > 0) {
#if _MSC_VER <= 1200  // VC6 or earlier.
    V = (unsigned)rand();
#else
    (void)rand_s(&V);
#endif
    for (L = SEGGER_MIN(DataLen, 4); L > 0; --L) {
      *pData = V & 0xFF;
      V >>= 8;
     ++pData;
     --DataLen;
    }
  }
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  fprintf(stderr, "CRYPTO: panic, system halted.\n");
  exit(100);
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  static const CRYPTO_RNG_API _RNG_Win32 = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,     NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SHA3_224_Install (&CRYPTO_HASH_SHA3_224_SW,   NULL);
  CRYPTO_SHA3_256_Install (&CRYPTO_HASH_SHA3_256_SW,   NULL);
  CRYPTO_SHA3_384_Install (&CRYPTO_HASH_SHA3_384_SW,   NULL);
  CRYPTO_SHA3_512_Install (&CRYPTO_HASH_SHA3_512_SW,   NULL);
  CRYPTO_SM3_Install      (&CRYPTO_HASH_SM3_SW,        NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_SW,      NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 using Win32 s_rand()
  // as an entropy source.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG_Win32);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}
```

```
/*************************** End of file ****************************/
```

6.4.3 Linux configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Linux:

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_Full_Linux.c
Purpose     : Configure full cryptography for Linux.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"
#include <stdlib.h>
#include <stdio.h>

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get entropy from /dev/urandom.
*
*  Parameters
*    pData   - Pointer to object that receives the random bitstream.
*    DataLen - Octet length of the object.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  static FILE *pFile;
  //
  pFile = fopen("/dev/urandom", "rb");
  if (pFile == NULL) {
    CRYPTO_X_Panic();
  }
  fread(pData, 1, DataLen, pFile);
  fclose(pFile);
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
```

```
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  fprintf(stderr, "CRYPTO: panic, system halted.\n");
  exit(100);
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  static const CRYPTO_RNG_API _RNG_Linux = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SHA3_224_Install (&CRYPTO_HASH_SHA3_224_SW,   NULL);
  CRYPTO_SHA3_256_Install (&CRYPTO_HASH_SHA3_256_SW,   NULL);
  CRYPTO_SHA3_384_Install (&CRYPTO_HASH_SHA3_384_SW,   NULL);
  CRYPTO_SHA3_512_Install (&CRYPTO_HASH_SHA3_512_SW,   NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_SW,      NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 with /dev/urandom as an entropy source.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG_Linux);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ****************************/
```

6.5 emCrypt API reference

The following sections are extracted from the full emCrypt documentation for reference.

6.5.1 API functions

Function	Description
Hashes	
CRYPTO_MD5_Install()	Install MD5 hash implementation.
CRYPTO_SHA1_Install()	Install SHA-1 hash implementation.
CRYPTO_SHA256_Install()	Install SHA-256 hash implementation.
CRYPTO_SHA512_Install()	Install SHA-512 hash implementation.
Ciphers	
CRYPTO_AES_Install()	Install cipher.
CRYPTO_TDES_Install()	Install cipher.
CRYPTO_SEED_Install()	Install cipher.
CRYPTO_ARIA_Install()	Install cipher.
CRYPTO_CAMELLIA_Install()	Install cipher.

6.5.1.1 CRYPTO_MD5_Install()

Description

Install MD5 hash implementation.

Prototype

```
void CRYPTO_MD5_Install(const CRYPTO_HASH_API * pHWAPI,
                        const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.2 CRYPTO_SHA1_Install()

Description

Install SHA-1 hash implementation.

Prototype

```
void CRYPTO_SHA1_Install(const CRYPTO_HASH_API * pHWAPI,
                         const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.3 CRYPTO_SHA256_Install()

Description

Install SHA-256 hash implementation.

Prototype

```
void CRYPTO_SHA256_Install(const CRYPTO_HASH_API * pHWAPI,
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.4 CRYPTO_SHA512_Install()

Description

Install SHA-512 hash implementation.

Prototype

```
void CRYPTO_SHA512_Install(const CRYPTO_HASH_API * pHWAPI,
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.5 CRYPTO_AES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_AES_Install(const CRYPTO_CIPHER_API * pHWAPI,
                        const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.6 CRYPTO_TDES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_TDES_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.7 CRYPTO_SEED_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_SEED_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.8 CRYPTO_ARIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_ARIA_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

6.5.1.9 CRYPTO_CAMELLIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_CAMELLIA_Install(const CRYPTO_CIPHER_API * pHWAPI,
                             const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

Chapter 7

OS Integration

7.1 SSL-OS integration

emSSL can be configured for use in a multitasking environment. The interface to the operating system is encapsulated in a single file and a number of standard integrations exist.

This chapter provides descriptions of the functions required to fully support emSSL in multitasking environments.

7.1.1 SSL-OS API

Function	Description
General functions	
`SSL_OS_Init()`	Initialize SSL stack.
`SSL_OS_Lock()`	Lock SSL stack.
`SSL_OS_Unlock()`	Unlock SSL stack.
Macros	
`SSL_OS_LOCK()`	A macro that locks the SSL library and prevents simultaneous use of critical resources. Typically this is defined as a call to `SSL_OS_Lock()` but may be defined in `SSL_Conf.h`.
`SSL_OS_UNLOCK()`	A macro that unlocks the previously-locked SSL library. Typically this is defined as a call to `SSL_OS_Unlock()` but may be defined in `SSL_Conf.h`.

7.1.1.1 SSL_OS_Init()

Description

Initialize SSL stack.

Prototype

```
void SSL_OS_Init(void);
```

Additional information

Creates and initializes all objects required for task synchronization.

7.1.1.2 SSL_OS_Lock()

Description

Lock SSL stack.

Prototype

```
void SSL_OS_Lock(void);
```

Additional information

The stack requires a single lock, typically a resource semaphore or mutex. This function locks this object, guarding sections of the stack code against other threads. If the entire stack executes from a single task, no functionality is required here.

It is required that the lock is “recursive” or “counts” and can be locked and unlocked several times by the same calling task.

7.1.1.3 SSL_OS_Unlock()

Description

Unlock SSL stack.

Prototype

```
void SSL_OS_Unlock(void);
```

Additional information

This function unlocks the single lock locked by a previous call to `SSL_OS_Lock()`. If the entire stack executes from a single task, no functionality is required here.

7.1.2 SSL-OS binding for embOS

The following is a sample binding for SEGGER embOS, `SSL_OS_embOS.c`:

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SSL_OS_embOS.c
Purpose : Kernel abstraction for embOS

*/

#include "SSL_Int.h"
#include "RTOS.h"

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8 _IsInited;

/*********************************************************************
*
*       Public data
*
**********************************************************************
*/

OS_RSEMA  SSL_OS_RSema;     // Public only to allow inlining
 (direct call from SSL core)

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       SSL_OS_Init
*
*  Function description
*    Initialize (create) all objects required for task synchronization.
*/
void SSL_OS_Init(void) {
  if (_IsInited == 0) {
    OS_CREATERSEMA(&SSL_OS_RSema);
    _IsInited = 1;
  }
}

/*********************************************************************
*
*       SSL_OS_DisableInterrupt
*/
void SSL_OS_DisableInterrupt(void) {
  OS_IncDI();
}
```

```
/*********************************************************************
*
*       SSL_OS_EnableInterrupt
*/
void SSL_OS_EnableInterrupt(void) {
  OS_DecRI();
}

/*********************************************************************
*
*       SSL_OS_Lock
*
*  Function description
*    The stack requires a single lock, typically a resource semaphore
*    or mutex. This function locks this object, guarding sections of
*    the stack code against other threads.
*    If the entire stack executes from a single task, no
*    functionality is required here.
*/
void SSL_OS_Lock(void) {
  OS_Use(&SSL_OS_RSema);
}

/*********************************************************************
*
*       SSL_OS_Unlock
*
*  Function description
*    Unlocks the single lock used locked by a previous call to
*    SSL_OS_Lock().
*/
void SSL_OS_Unlock(void) {
  OS_Unuse(&SSL_OS_RSema);
}

/*********************************************************************
*
*       SSL_OS_GetTime32()
*
*  Function description
*    Return the current system time in ms.
*    The value will wrap around after app. 49.7 days. This is taken
*    into account by the stack.
*/
U32 SSL_OS_GetTime32(void) {
  return OS_GetTime32();
}

/*********************************************************************
*
*       SSL_OS_GetTaskName()
*
* Function description
*   Retrieves the task name (if available from the OS and not in
*   interrupt) for the currently active task.
*
* Parameters
*   pTask: Pointer to a task identifier such as a task control block.
*
* Return value
*   Terminated string with task name.
*/
const char * SSL_OS_GetTaskName(void *pTask) {
  return OS_GetTaskName((OS_TASK*)pTask);
}
```

```
/*************************** End of file ****************************/
```

7.1.3 SSL-OS binding for bare metal

The following is a sample binding for a bare metal system that has no tasking, `SSL_OS_None.c`:

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SSL_OS_None.c
Purpose : Kernel abstraction for usage of emSSL without any RTOS.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSL_Int.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       SSL_OS_Init()
*
*  Function description
*    Initialize SSL stack.
*
*  Additional information
*    Creates and initializes all objects required for task
*    synchronization.
*/
void SSL_OS_Init(void) {
  /* Empty */
}

/*********************************************************************
*
*       SSL_OS_DisableInterrupt
*/
void SSL_OS_DisableInterrupt(void) {
  /* Empty */
}

/*********************************************************************
*
*       SSL_OS_EnableInterrupt
*/
void SSL_OS_EnableInterrupt(void) {
  /* Empty */
}

/*********************************************************************
*
```

```
*       SSL_OS_Lock()
*
*  Function description
*    Lock SSL stack.
*
*  Additional information
*    The stack requires a single lock, typically a resource semaphore
*    or mutex. This function locks this object, guarding sections of
*    the stack code against other threads. If the entire stack
*    executes from a single task, no functionality is required here.
*
*    It is required that the lock is "recursive" or "counts" and
*    can be locked and unlocked several times by the same calling task.
*/
void SSL_OS_Lock(void) {
  /* Empty */
}

/*********************************************************************
*
*       SSL_OS_Unlock()
*
*  Function description
*    Unlock SSL stack.
*
*  Additional information
*    This function unlocks the single lock locked by a previous call
*    to SSL_OS_Lock().  If the entire stack executes from a single
*    task, no functionality is required here.
*/
void SSL_OS_Unlock(void) {
  /* Empty */
}

/*********************************************************************
*
*       SSL_OS_GetTime32()
*
*  Function description
*    Return the current system time in ms.
*    The value will wrap around after app. 49.7 days.
*/
U32 SSL_OS_GetTime32(void) {
  return 0;
}

/*********************************************************************
*
*       SSL_OS_GetTaskName()
*
*  Function description
*    Get task name.
*
*  Parameters
*    pTask - Pointer to a task identifier such as a task control block.
*
*  Return value
*    Terminated string with task name.
*
*  Additional information
*   Retrieves the task name (if available from the OS and not in
*   an interrupt) for the currently active task.
*/
const char * SSL_OS_GetTaskName(void *pTask) {
  SSL_USE_PARA(pTask);  // Avoid warning 'parameter
 "pTask" was never referenced'.
  return "emSSL";
}
```

```
/*************************** End of file ****************************/
```

7.2 CRYPTO-OS integration

In a threaded execution environment individual hardware resources must be protected from simultaneous use by more than one thread. emSSL does this by surrounding use of hardware resources by calls to an OS binding layer.

To use a shared resource, emSSL will either:

- Call `CRYPTO_OS_Claim()`, use the resource, and call `CRYPTO_OS_Unclaim()` to release it, or
- Call `CRYPTO_OS_Request()` to request access to the resource. If access is granted, emSSL uses the resource and then calls `CRYPTO_OS_Unclaim()` to release it. In the case where access to the resource is not granted, emSSL will not use the resource and will not call `CRYPTO_OS_Unclaim()`.

The parameter `Unit` is a zero-based index to the hardware being requested and is defined by the specific hardware platform or target device that is in use. No hardware acceleration interface in emSSL requires more than three units (e.g. a ciphering unit, a hashing unit, and a random number generation unit). The specific requirements for each device are described in the relevant sections.

As an OS layer may well need to create mutexes or semaphores corresponding to each unit, `CRYPTO_OS_Init()` is called as part of emSSL initialization.

7.2.1 CRYPTO-OS API

Function	Description
`CRYPTO_OS_Init()`	Initialize CRYPTO binding to OS.
`CRYPTO_OS_Claim()`	Claim a hardware resource.
`CRYPTO_OS_Request()`	Test-and-claim a hardware resource.
`CRYPTO_OS_Unclaim()`	Release claim on a hardware resource.

7.2.1.1 CRYPTO_OS_Init()

Description

Initialize CRYPTO binding to OS.

Prototype

```
void CRYPTO_OS_Init(void);
```

Additional information

This function should initialize any semaphores or mutexes used for protecting each hardware unit.

7.2.1.2 CRYPTO_OS_Claim()

Description

Claim a hardware resource.

Prototype

```
void CRYPTO_OS_Claim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Claim the hardware resource that corresponds to the unit index. In a threaded environment, this function should block a task requesting a resource that is already in use by using a semaphore or mutex, for example. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function can be empty.

7.2.1.3 CRYPTO_OS_Request()

Description

Test-and-claim a hardware resource.

Prototype

```
int CRYPTO_OS_Request(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Return value

= 0	Resource is already in use and was not claimed.
≠ 0	Resource claimed.

Additional information

Attempt to claim the hardware resource that corresponds to the unit index. In a threaded environment, this function is a nonblocking test-and-lock of a semaphore or mutex. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function should always return nonzero, i.e. resource claimed.

7.2.1.4 CRYPTO_OS_Unclaim()

Description

Release claim on a hardware resource.

Prototype

```
void CRYPTO_OS_Unclaim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Release the claim the hardware resource that corresponds to the unit index. This will only be called to unclaim a claimed resource.

7.2.2 CRYPTO-OS binding for embOS

The following is a sample binding for SEGGER embOS, `CRYPTO_OS_embOS.c`:

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_OS_embOS.c
Purpose     : SEGGER embOS CRYPTO-OS binding.

*/

/*********************************************************************
*
*       #include section
*
**********************************************************************
*/

#include "CRYPTO_Int.h"
#include "RTOS.h"

/*********************************************************************
*
*       Preprocessor definitions, configurable
*
**********************************************************************
*/

#ifndef   CRYPTO_CONFIG_OS_MAX_UNIT
  #define CRYPTO_CONFIG_OS_MAX_UNIT     (CRYPTO_OS_MAX_INTERNAL_UNIT + 3)
#endif

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static OS_RSEMA _aSema[CRYPTO_CONFIG_OS_MAX_UNIT];

/*********************************************************************
*
*       Public functions
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_OS_Claim()
*
*  Function description
*    Claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Claim(unsigned Unit) {
  CRYPTO_ASSERT(Unit < CRYPTO_CONFIG_OS_MAX_UNIT);
  //
  OS_Use(&_aSema[Unit]);
}

/*********************************************************************
*
*       CRYPTO_OS_Request()
*
*  Function description
*    Request a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*
*  Return value
*    == 0 - Resource is already in use and was not claimed.
*    != 0 - Resource claimed.
*/
int CRYPTO_OS_Request(unsigned Unit) {
  CRYPTO_ASSERT(Unit < CRYPTO_CONFIG_OS_MAX_UNIT);
  //
  return OS_Request(&_aSema[Unit]);
}

/*********************************************************************
*
*       CRYPTO_OS_Unclaim()
*
*  Function description
*    Release claim on a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
  CRYPTO_ASSERT(Unit < CRYPTO_CONFIG_OS_MAX_UNIT);
  //
  OS_Unuse(&_aSema[Unit]);
}

/*********************************************************************
*
*       CRYPTO_OS_Init()
*
*  Function description
*    Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
  unsigned Unit;
  //
  for (Unit = 0; Unit < CRYPTO_CONFIG_OS_MAX_UNIT; ++Unit) {
    OS_CreateRSema(&_aSema[Unit]);
  }
}

/*************************** End of file ****************************/
```

7.2.3 CRYPTO-OS binding for bare metal

The following is a sample binding for a bare metal system that has no tasking, `CRYPTO_OS_None.c`:

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_OS_None.c
Purpose     : Bare metal CRYPTO-OS binding.

*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_OS_Claim()
*
*  Function description
*    Claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Claim(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
}

/*********************************************************************
*
*       CRYPTO_OS_Request()
*
*  Function description
*    Test-and-claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*
*  Return value
*    == 0 - Resource is already in use and was not claimed.
*    != 0 - Resource claimed.
*/
int CRYPTO_OS_Request(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
  return 1;
}

/*********************************************************************
*
*       CRYPTO_OS_Unclaim()
*
*  Function description
*    Release claim on a hardware resource.
*
```

```
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
}

/*********************************************************************
*
*       CRYPTO_OS_Init()
*
*  Function description
*    Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
  /* Nothing to do. */
}

/*************************** End of file ****************************/
```

Chapter 8

Resource usage

This chapter covers the resource usage of emSSL. It contains information about the memory requirements in typical systems, which can be used to obtain sufficient estimates for most target systems.

8.1 Memory footprint

emSSL is designed to cater for many different embedded design requirements, from constrained microcontrollers to high performance microprocessors. Some features might be excluded from a build in order to construct a highly compact, minimal system. Note that the values are only valid for the given configuration.

8.1.1 Target system configuration

The following table shows the hardware and the toolchain details of a typical emSSL target system:

Detail	Description
CPU	Cortex-M4
Tool chain	SEGGER Embedded Studio for ARM v3.30
Model	Thumb-2 instructions
Compiler options	Highest size optimization

8.1.2 ROM use

The following table indicates the ROM requirement for each of emSSL's components:

Component	Size (approximate)
Public key algorithms	
ECDSA	0.4 KB
RSA-PKCS1	0.5 KB
Hash and MAC functions	
SHA-1	0.5 KB
SHA-256 (including SHA-224)	0.9 KB
SHA-512 (including SHA-384)	1.9 KB
MD5	0.8 KB
HMAC-SHA1	0.2 KB
HMAC-SHA256	0.2 KB
HMAC-SHA384	0.2 KB
Cipher functions	
DES	3.7 KB
AES	3.4 KB
AES-GCM (requires AES)	0.5 KB
ARIA	3.1 KB
SEED	1.1 KB
Camellia	3.7 KB
Protocol support	
TLS core client and server combined	7.3 KB (TLS 1.0 through 1.2)
Cipher suites	1.8 KB (for all suites)
X.509 support	2.2 KB
PRF-TLS1	0.4 KB
PRF-SHA256	0.2 KB
PRF-SHA384	0.2 KB
Shared supporting code	

Component	Size (approximate)
MPI for RSA and ECC support	4.5 KB
Curve storage	4.4 KB (for all curves)
Curve arithmetic (requires MPI)	2.3 KB
Memory management	0.3 KB

Typical configurations

The following table lists approximate sizes for simple configurations with software implementations of all algorithms, i.e. no hardware acceleration, compiled for minimum size, no mutual authentication, with a single cipher suite and P-256 curve installed.

Configuration	ROM (CRYPTO)	ROM (SSL)	Total
Server mode			
RSA-AES-CBC-SHA256	9.9 KB	7.1 KB	17.0 KB
ECDHE-RSA-AES-128-CBC-SHA256	15.3 KB	7.5 KB	22.8 KB
ECDHE-ECDSA-AES-128-CBC-SHA256	15.2 KB	7.5 KB	22.7 KB
Client mode			
RSA-AES-CBC-SHA256	13.2 KB	9.5 KB	22.7 KB
ECDHE-RSA-AES-128-CBC-SHA256	17.2 KB	9.9 KB	27.1 KB
ECDHE-ECDSA-AES-128-CBC-SHA256	16.9 KB	9.6 KB	26.5 KB

8.1.3 RAM use

emSSL’s RAM use can be partitioned as follows:

- *Static data requirement* — a fixed overhead incurred by using emSSL.
- *State and key material* — a variable overhead per connection that stores connection state and any key material required by the connection.
- *Public key memory* — a variable overhead for carrying out public key algorithms when negotiating a connection.
- *Protocol memory* — memory required to store protocol packets before encryption or decryption.

8.1.3.1 Static overhead

emSSL requires approximately 0.5 KB of RAM as a fixed overhead to manage its operation.

8.1.3.2 Connection state and key material

RAM is required to store the state of each active SSL connection. Part of this requirement is a variable amount of state that depends upon the cipher suite negotiated between the peers.

- Each SSL session object requires approximately 700 bytes to store its state, excluding cipher suite state.
- Additional memory for cipher suite state varies with cipher suite, but typically about 500 bytes is used.

8.1.3.3 Public key algorithms

Temporary memory is required to run appropriate public key algorithms when keys are exchanged. The amount of memory requires depends upon the public key algorithm and the key sizes. However, typically P-256 curves and 2048-bit RSA public keys require approximately 5 KB of memory to run.

8.1.3.4 Protocol memory

The number of bytes required per connection depends upon the packet size delivered from a server to a client. Each record layer fragment can be up to 16 KB in size. As a client, you have no control over the size of the fragment sent by a server and a TLS client is expected to deal with these large fragments.

As a server the amount of memory required is limited by the data that is sent; the TLS record layer will encapsulate any data sent into packets, so application code is completely in control of what is sent to the peer and its ultimate size.

8.1.3.5 Calculating RAM requirements

The maximum used amount has to be provided in `SSL_X_Config()` while initializing emSSL and has to be available until closing emSSL.

The overall RAM requirements can be approximated if you know the maximum number of connections that are made at the same time in your application.

```
( (700 Byte + 500 Byte) + (2 * 16 kByte) ) * NumConnections + 0.5 kByte
```

If you allow a maximum of 2 connections the RAM requirements are:

```
( (700 Byte + 500 Byte) + (2 * 16 kByte) ) * 2 + 0.5 kByte = 66.9 kByte
```

8.2 Minimal system configuration

emSSL can be configured to match nearly every system requirements.

A minimal client configuration with a single cipher suite requires less than 25 KB of RAM. A maximum total of 33.2 KB RAM is required to handle a single connection.

Chapter 9

Best practice

This chapter provides some guidance on how to configure your client and server in order to meet the advice given in RFC 7525, *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*.

Meeting the requirements of RFC 7527 in the following sections is generally accomplished, and completely localized without changing any source code, by setting up emSSL correctly in the `SSL_X_Config()` function.

9.1 SSL/TLS protocol versions

This corresponds to section 3.1.1 of RFC 7525.

Requirement: *Implementations MUST NOT negotiate SSL version 2.*

emSSL does not support SSL version 2 and no configuration is necessary to satisfy this requirement.

Requirement: *Implementations MUST NOT negotiate SSL version 3.*

emSSL does not support SSL version 3 and no configuration is necessary to satisfy this requirement.

Requirement: *Implementations SHOULD NOT negotiate TLS version 1.0.*

You can disable support for TLS version 1.0 by not adding the TLS 1.0 protocol when configuring emSSL. That is, do not call `SSL_PROTOCOL_Add()` specifying `SSL_PROTOCOL_TLS1v0_API` in `SSL_X_Config()`.

Requirement: *Implementations SHOULD NOT negotiate TLS version 1.1.*

You can disable support for TLS version 1.1 by not adding the TLS 1.1 protocol when configuring emSSL. That is, do not call `SSL_PROTOCOL_Add()` specifying `SSL_PROTOCOL_TLS1v1_API` in `SSL_X_Config()`.

Requirement: *Implementations MUST support TLS version 1.2 and MUST prefer to negotiate TLS version 1.2 over earlier versions of TLS.*

You can configure support for TLS version 1.2 by calling `SSL_PROTOCOL_Add()` specifying `SSL_PROTOCOL_TLS1v2_API` in `SSL_X_Config()` specifying the TLS 1.2 protocol. emSSL ensures that the latest protocols are negotiated before falling back to earlier protocols (in the case that multiple protocols have been added to emSSL). If only the TLS version 1.2 protocol has been added to emSSL, TLS will only negotiate a TLS 1.2 connection with TLS 1.2 cipher suites and will not fall back to any earlier protocol.

9.2 DTLS protocol versions

This corresponds to section 3.1.2 of RFC 7525.

emSSL does not support DTLS in this version and therefore these requirements are not relevant.

9.3 Fallback to lower versions

This corresponds to section 3.1.3 of RFC 7525.

Requirement: *Clients that "fall back" to lower versions of the protocol after the server rejects higher versions of the protocol MUST NOT fall back to SSLv3 or earlier.*

emSSL does not support SSL version 3 and earlier, and no configuration is necessary to satisfy this requirement.

9.4 Compression

This corresponds to section 3.3 of RFC 7525.

Requirement: *In order to help prevent compression-related attacks (summarized in Section 2.6 of RFC 7457), implementations and deployments SHOULD disable TLS-level compression (Section 6.2.2 of [RFC5246]), unless the application protocol in question has been shown not to be open to such attacks.*

emSSL does not support compression and no configuration is necessary to satisfy this requirement.

9.5 Server name indication

This corresponds to section 3.6 of RFC 7525.

Requirement: *TLS implementations MUST support the Server Name Indication (SNI) extension defined in Section 3 of [RFC6066] for those higher-level protocols that would benefit from it, including HTTPS.*

emSSL will ensure that the SNI extension is added if a server name is provided when connecting using `SSL_SESSION_Connect()`.

9.6 General guidelines

This corresponds to section 4.1 of RFC 7525.

Requirement: *Implementations MUST NOT negotiate the cipher suites with `NULL` encryption.*

You can disable support of `NULL` ciphers by not adding null cipher suites when configuring emSSL. That is, do not call `SSL_SUITE_Add()` in `SSL_X_Config()` with a suite that specifies null encryption.

Requirement: *Implementations MUST NOT negotiate RC4 cipher suites.*

You can disable support of RC4 cipher suites by not adding RC4 cipher suites when configuring emSSL. That is, do not call `SSL_SUITE_Add()` in `SSL_X_Config()` with a suite that specifies RC4 encryption.

Requirement: *Implementations MUST NOT negotiate cipher suites offering less than 112 bits of security, including so-called "export-level" encryption (which provide 40 or 56 bits of security).*

You can disable support of weaker cipher suites by not adding DES (as opposed to TDES) cipher suites when configuring emSSL. That is, do not call `SSL_SUITE_Add()` in `SSL_X_Config()` with a suite that specifies DES encryption.

Requirement: *Implementations SHOULD NOT negotiate cipher suites that use algorithms offering less than 128 bits of security.*

You can disable support of weaker cipher suites by not adding DES / TDES cipher suites when configuring emSSL. That is, do not call `SSL_SUITE_Add()` in `SSL_X_Config()` with a suite that specifies DES or 3DES encryption.

Requirement: *Implementations SHOULD NOT negotiate cipher suites based on RSA key transport, a.k.a. "static RSA".*

You can disable support of static RSA cipher suites by not adding the static RSA cipher suites when configuring emSSL. That is, do not call `SSL_SUITE_Add()` in `SSL_X_Config()` with a suite that specifies plain RSA key transport (such as `SSL_SUITE_RSA_WITH_AES_256_CBC_SHA`).

Many sites only offer static RSA cipher suites, for instance `www.apple.com`. You must enable support for static RSA cipher suites if you wish to connect to these sites with TLS.

Requirement: *Implementations MUST support and prefer to negotiate cipher suites offering forward secrecy, such as those in the Ephemeral Diffie-Hellman and Elliptic Curve Ephemeral Diffie-Hellman ("DHE" and "ECDHE") families.*

emSSL supports both DHE and ECDHE cipher suites for TLS version 1.2. The user is responsible for setting the cipher suite preference for emSSL clients and servers during emSSL initialization. The order that cipher suites are added to emSSL is the order in which clients offer them to servers, and the order that servers will select a suite offered by a client. In order to satisfy this requirement, add any DHE and ECDHE cipher suites you wish to support before any other cipher suite.

9.7 Recommended cipher suites

This corresponds to section 4.2 in RFC 7525.

Requirement: *Given the foregoing considerations, implementation and deployment of the following cipher suites is RECOMMENDED: TLS-DHE-RSA-WITH-AES-128-GCM-SHA256, TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256, TLS-DHE-RSA-WITH-AES-256-GCM-SHA384, TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384.*

emSSL supports all four named cipher suites.

Requirement: *Typically, in order to prefer these suites, the order of suites needs to be explicitly configured in server software.*

You can deploy these cipher suites by adding each of them (or a subset) to emSSL using `SSL_SUITE_Add()` in `SSL_X_Config()` with the usual convention that the order of suite addition defines the order of preference for client and server.

9.8 Implementation details

This corresponds to section 4.2.1 in RFC 7525.

Requirement: *Clients SHOULD include TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256 as the first proposal to any server, unless they have prior knowledge that the server cannot respond to a TLS 1.2 `client_hello` message.*

To satisfy this requirement, add the suite to emSSL first using `SSL_SUITE_Add()` in `SSL_X_Config()`.

Requirement: *To maximize interoperability, RFC 5246 mandates implementation of the TLS-RSA-WITH-AES-128-CBC-SHA cipher suite, which is significantly weaker than the cipher suites recommended here.*

If you wish to comply with RFC 5246, add this suite to emSSL using `SSL_SUITE_Add()` in `SSL_X_Config()`.

Requirement: *Note that some profiles of TLS 1.2 use different cipher suites. For example, [RFC6460] defines a profile that uses the TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256 and TLS-ECDHE-ECDSA-WITH-AES-256-GCM-SHA384 cipher suites.*

If you wish to comply with RFC 6460, for example, add the appropriate suites to emSSL using `SSL_SUITE_Add()` in `SSL_X_Config()`.

Requirement: *Both clients and servers SHOULD include the "Supported Elliptic Curves" extension [RFC4492].*

emSSL supports this extension by default and automatically includes the extension when negotiating cipher suites that require elliptic curve cryptography.

Requirement: *For interoperability, clients and servers SHOULD support the NIST P-256 (secp256r1) curve [RFC4492].*

emSSL supports all NIST prime curves. In order to configure emSSL for this curve, call `SSL_CURVE_Add(&SSL_CURVE_secp256r1)` in `SSL_X_Config()`.

Requirement: *In addition, clients SHOULD send an `ec_point_formats` extension with a single element, "uncompressed".*

emSSL only supports uncompressed points and automatically adds this extension when negotiating suites that require elliptic curve cryptography.

9.9 Truncated HMAC

This corresponds to section 4.5 in RFC 7525.

Requirement: *Implementations MUST NOT use the Truncated HMAC extension, defined in Section 7 of [RFC6066].*

emSSL does not support the truncated HMAC extension and will never offer nor honor it.

Chapter 10

Reference information

The SSL and TLS protocols are not defined by a single specification but as a range of specifications maintained by the Internet Engineering Task Force (IETF) as Requests for Comments (RFCs). All RFCs are made available online at the IETF website.

This section collates reference information relating to the implementation of SSL and TLS protocols by emSSL and should be considered the definitive specification when understanding emSSL's behavior. In addition to the RFCs mentioned above, reference is also made to national standards documents for many cryptographic algorithms. The references cited here does not constitute an exhaustive collection because it omits X.509 certificate specifications and the various ASN.1 standards.

Because information relating to the various TLS protocols, cipher suites, and extensions is covered by so many standards documents, it is may be that we have not correctly understood or implemented all features of emSSL according to these standards. If you believe that emSSL does not comply with the appropriate standards, please contact us so we can investigate any discrepancy where you believe we fall short.

10.1 Cipher suites

Cipher suites are needed to establish a secure connection with TLS. In the negotiation phase of the connection establishment the client and the server must agree one one cipher suite to use for the communication.

It is not required to support all cipher suites which are defined for TLS, but as a client it is important to support the most common cipher suites to be able to communicate with any server.

A cipher suite is a combination of four components:

- The authentication algorithm, authenticating the message provider (the server)
- The key exchange algorithm, used to exchange the necessary keys for the communication
- The encryption ciphers, used to bulk encrypt the actual data with the derived session keys
- A message authentication code (MAC), used to safely authenticate the message.

10.2 TLS specifications

emSSL implements TLS versions 1.0 through 1.2, but does not implement SSLv2 and SSLv3 by design as these are known to be insecure.

The base TLS specifications are spread across three different RFCs for the three supported versions:

The TLS Protocol Version 1.0 (RFC 2246)

http://tools.ietf.org/html/rfc2246

The Transport Layer Security (TLS) Protocol Version 1.1 (RFC 4346)

http://tools.ietf.org/html/rfc4346

The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)

http://tools.ietf.org/html/rfc5246

10.3 Extensions

The base documents define the underlying protocol and evolve slowly. Extensions to TLS, which are optional, are proposed and ratified independently. The following RFCs define extensions to TLS.

Transport Layer Security (TLS) Extensions (RFC 3546)

http://tools.ietf.org/html/rfc3546

Pre-Shared Key Ciphersuites for Transport Layer Security (RFC 4279)

http://tools.ietf.org/html/rfc4279

Transport Layer Security (TLS) Extensions: Extension Definitions (RFC 6066)

http://tools.ietf.org/html/rfc6066

Transport Layer Security (TLS) Authorization Extensions (RFC 5878)

http://tools.ietf.org/html/rfc5878

Transport Layer Security (TLS) Renegotiation Indication Extension (RFC 5746)

http://tools.ietf.org/html/rfc5746

Prohibiting Secure Sockets Layer (SSL) Version 2.0 (RFC 6176)

http://tools.ietf.org/html/rfc6176

TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks

http://tools.ietf.org/html/draft-ietf-tls-downgrade-scsv-00

10.4 Cryptography

These documents describe the underlying cryptographic primitives that TLS and SSL require. In fact, SSL and TLS mandate only one supported cipher suite per version, but emSSL implements many more cipher suites so that you have flexibility to decide which suites to select for the security you require and also to provide a broad range of connection options to enhance client and server compatibility.

Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 (RFC 3447)

http://www.ietf.org/rfc/rfc3447.txt

AES-CCM Cipher Suites for Transport Layer Security (RFC 6655)

http://tools.ietf.org/html/rfc6655

AES Galois Counter Mode (GCM) Cipher Suites for TLS (RFC 5288)

http://tools.ietf.org/html/rfc5288

Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (RFC 4492)

http://tools.ietf.org/html/rfc4492

Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (RFC 7027)

http://tools.ietf.org/html/rfc7027

Elliptic Curve Cryptography Subject Public Key Information (RFC 5480)

http://tools.ietf.org/html/rfc5480

Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality (SP 800-38C)

This is a NIST specification rather than an RFC.

http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf

FIPS PUB 186-4 Digital Signature Standard (DSS)

http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

Mathematical routines for the NIST prime elliptic curves

https://www.nsa.gov/ia/_files/nist-routines.pdf

10.5 Hardware acceleration

10.5.1 RSA and elliptic curve accelerators

Using a modular arithmetic accelerator will speed up RSA and elliptic curve operations over prime fields which reduces key exchange time considerably. Although emSSL contains a robust implementation of modular exponentiation, it is no match for hardware acceleration.

Unfortunately, devices that offer strong cryptography are subject to export controls. Worse still, many vendors restrict the documentation for both the devices themselves and the cryptographic accelerators they implement. Executing a non-disclosure agreement (NDA) with the silicon vendor to access the information and in order to provide support in emSSL still prevents delivering that implementation to customers unless the customer also has a non-disclosure agreement with the silicon vendor.

For example, the following devices have cryptographic modular arithmetic accelerators but their cryptographic abilities are covered by an NDA:

- Atmel SAMA5D4
- Maxim MAX32550
- Maxim MAXQ1103 and MAXQ1850

However, some devices are sufficiently open:

- Silicon Labs EFM32 (Pearl and Jade Gecko)

10.5.2 Hash and MAC accelerators

Hash and MAC algorithms cannot be used for encryption and, as such, export controls usually do not apply to them. Many modern devices that offer Ethernet hardware also offer a SHA accelerator in full expectation that implementations will make use of it.

Whilst the SHA accelerator will speed up the bulk encipherment process after a connection is established, it does little for the key agreement phase of a connection beyond speeding up signature generation and verification.

10.5.3 Bulk encipherment accelerators

The same devices that offer hash and MAC capabilities usually offer bulk encipherment with an AES accelerator for AES-128 or AES-256, or perhaps both. And fortunately, it seems that device datasheets and reference manuals that document how the particular AES accelerator functions are not under NDA.

Whilst the AES accelerator will speed up the bulk encipherment process after a connection is established, it does nothing for the key agreement phase of a connection or for signature generation and verification.

10.5.4 Vendor-optimized and certified libraries

It may well be that you would like to use vendor-optimized libraries provided for your device, or even NIST-certified “cryptograpic components” (i.e. libraries). It is possible to swap out parts of emSSL’s cryptography and replace it with another implementation, but doing so is beyond the scope of this document.

Because there is no standard cryptography API, integrating alternative hardware and software libraries will require some effort. Please see the next section for how to proceed.

10.5.5 What to do if you require alternative cryptography

If you would like to replace any part of the standard emSSL cryptography implementation with either a hardware-accelerated or certified implementation that we do not already support, please contact us. It may well be that we have a particular hardware accelerator already implemented for emSSL and ready to go, or we may have already written the support code to transition from emSSL’s APIs to other vendor library APIs.

Chapter 11

Patents and export controls

This section tries to describe the patents and licenses you may require to deploy SSL *in general*, whatever SSL solution you choose, be it emSSL or some other product. It also describes export controls that may apply to your equipment.

Patents are granted, challenged, and struck down over time, in different geographical regions, and for different fields of use; these facts alone make it impossible to provide a factually-accurate, blanket statement regarding all end-customer equipment. Export controls change in the same way, but usually at a slower pace.

Export controls apply to emSSL itself and the end user equipment in order to restrict the export of strong cryptography. emSSL uses strong cryptography for signing and encryption. And as export controls depend upon what you are exporting, what the purpose of the cryptographic device is, and what type of cryptography your device has, it is impossible to provide statements that cover all situations and devices.

Note

We strongly advise you to conduct your own research and consult legal advisors on deployment of SSL (and ECC cipher suites in particular) in your devices.

This content of this section is provided without warranty of any kind. It is your sole responsibility to decide whether or not you wish to make use of ECC technology in your product.

To the best of our knowledge, having researched the issue, the following sections are guidelines for the deployment of emSSL in end-user equipment.

11.1 Using RSA cipher suites

The patents relating to RSA cipher suites that use RSA signatures are now all expired and, to the best of our knowledge, the use of static RSA cipher suites requires no license from any patent holder.

11.2 Using DSA cipher suites

DSA cipher suites are not in common use. Despite this, the patents relating to DSA cipher suites are all expired and, to the best of our knowledge, the use of any DSA cipher suite requires no license from any patent holder.

Note

DSA patents were assigned to United States of America and appropriate patents were made available worldwide on a royalty-free basis.

11.3 Using ECC cipher suites

ECC is the most problematic public key cryptosystem because of active patents assigned to Certicom (at the time of writing, October 10, 2018). There is no concise, clear statement relating to the implementation of elliptic curve cryptography from Certicom, nor is there a simple means of discovering whether an implementation scheme is covered by an ECC patent—you will need to conduct your own patent search in your geographic region.

To the best of our knowledge, the functions provided by emSSL do not infringe on any *implementation* patent.

It is required that anybody contributing to an IETF standard document disclose all IPR that they hold relating to the standard. To deploy ECC cipher suites in your product, you may require a royalty-free license from Certicom:

https://www.certicom.com/images/pdfs/certicom%20-ipr-contribution-to-ietfsept08.pdf

In order to avoid this, *do not add ECC cipher suites* and *do not add ECDSA signature verification* when initializing emSSL. In this manner, no ECC code is linked into your application.

We believe that the specialized reducers are not covered by an implementation patent because of prior art. If this situation causes you concern, you can avoid deploying the specialized reducers in your code and use only the slower, simple reduction scheme that use algorithms from antiquity.

Note

We stress again that patents are issued covering both different geographic domains and fields of use. It is your responsibility to ensure that your end products (that contain emSSL) do not infringe patents in the locations that you produce and sell that equipment.

This contents of this section is provided without warranty of any kind. It is your sole responsibility to decide whether or not you wish to make use of ECC technology in your product and to seek independent legal advice.

11.4 Export controls

Strong cryptography is subject to export controls from many countries. Because emSSL supports strong cryptography and does not limit public key lengths, you must ensure that the equipment that you export complies with export controls in the country you export from and the country that you export into.

The European Union has a common dual-use goods list (including encryption items in Category 5, Part 2 “Information Security”) defined by EC Regulation No 428/2009. Several member states have, in addition, regulations concerning the import, supply, use or export of encryption items. emSSL has been provided to you in accordance with the EC regulations and national laws of Germany. Any export or transfer of the software with a destination outside the European Union requires export permission.

The choice of cipher suites and public key algorithms is completely configurable in emSSL, allowing you to tailor the cryptographic capability of the device deployed with emSSL.

Note

We stress again that it is impossible to provide any warranties of statements made regarding export controls. It is your responsibility to ensure that your end products (that contain emSSL) conform to the export controls in place at the time and place of export.

This content of this section is provided without warranty of any kind. It is your sole responsibility to decide whether your product with cryptographic capability complies with appropriate export controls and to seek independent legal advice.

Chapter 12

Glossary

3DES
: *Triple DES.* A classical means to extend the 56-bit key space of DES to 112 bits by combining two 56-bit keys in three DES operations (two-key 3DES-EDE). 3DES is also known as TDES in standards documentation.

AEAD
: *Authenticated Encryption with Additional Data.* A modern cipher mode that combines encryption with authentication where both can run in parallel and enhance throughput in hardware implementations. AES-GCM and AES-CCM are AEAD ciphers..

AES
: *Advanced Encryption Standard.* A modern 128-bit block cipher, specified by NIST, that replaces the DES standard.

ASN.1
: *Abstract Syntax Notation 1.* A specification of how to encode primitive data as octet streams.

CBC
: *Cipher Block Chaining.* A cipher mode that uses the output of the previous block as an input to the following block to be encrypted.

DES
: *Data Encryption Standard.* A retired 64-bit block cipher with 56-bit keys defined by NIST.

DH
: *Diffie-Hellman.* A key agreement scheme based on discrete logarithm cryptography.

DHE
: *Ephemeral Diffie-Hellman.* A key agreement scheme based on discrete logarithm cryptography where keys are generated once per connection and are unique for each connection. This guarantees Perfect Forward Secrecy (PFS).

DRBG
: *Deterministic Random Bit Generator.* A random bit generator that will generate the same sequence of bits given the same seed as input, but the output is random using standard randomness tests.

DSA
: *Digital Signature Algorithm.* The algorithm that signs a piece of data, specified in the Digital Signature Standard.

DSS
: *Digital Signature Standard.* The NIST digital signature standard that specifies the Digital Signature Algorithm (DSA).

DTLS
Datagram Transport Layer Security. A scheme similar to TLS that transports TLS messages over UDP datagrams.

ECB
Electronic Code Book. An insecure mode for a block cipher where each block is encrypted in isolation and not chained.

ECC
Elliptic Curve Cryptography. Cryptography based on elliptic curves.

ECDH
Elliptic Curve Diffie-Hellman. The equivalent of Diffie-Hellman using elliptic curves.

ECDHE
Elliptic Curve Diffie-Hellman Ephemeral. As ECDH but using ephemeral keys. Provides perfect forward secrecy (PFS).

ECDSA
Elliptic Curve Digital Signature Algorithm. A standard for digital signatures signed using elliptic curve cryptography rather than discrete log cryptography. The elliptic curve analog of the discrete log signature scheme (DSA).

FIPS
Federal Information Processing Standard. A standard issued by NIST for Federal use and widely adopted throughout the world.

GCM
Galois Counter Mode. A modern mode for a block cipher where the authentication tag is computed using arithmetic in a Galois field, GF(2^128)..

HMAC
Hashed Message Authentication Code. A MAC that is computed using a cryptographic hash function in combination with a secret key.

IANA
Internet Assigned Numbers Authority. IANA is responsible for the global coordination of the Internet protocol resources for TLS as part of its mandate.

IETF
Internet Engineering Task Force. The IETF produces high quality, relevant technical documents that influence the way people design, use, and manage the Internet.

MAC
Message Authentication Code. A small piece of information used to authenticate a message and to provide integrity and authenticity assurances about the content of the message..

MD5
Message Digest Algorithm 5. A MAC defined by RSA Data Security, Inc.

MPI
Multiprecision integer. An integer that can grow and shrink as required to represent cryptographic numbers.

NIST
National Institute of Standards and Technology. An organization in the USA responsible for the standardization of a wide range of technologies that pervade the IT industry.

PFS
Perfect Forward Secrecy. A means to ensure that exposure of the session keys for one connection and its decryption does not expose other sessions to subsequent decryption using the recovered cryptographic material.

PKI
Public Key Infrastructure. A set of specifications and mechanisms that can provide confidence in and interoperability of public key cryptography systems.

PRF
Pseudorandom Function. A function defined in the TLS specifications used to generate various unpredictable internal data used by TLS connections.

PSK
: *Preshared Key.* A shared private key held by two entities agreed in advance of communication.

RFC
: *Request For Comment.* The standard means that IETF disseminates Internet standards.

RNG
: *Random Number Generator.* A device that generates true random numbers.

RSA
: *Rivest, Shamir, Adleman.* The name of the cryptosystem based on Integer Factorization problems defined by the three authors.

SHA
: *Secure Hash Algorithm.* The standard set of one-way functions that provide a message digest, as specified by NIST.

SSL
: *Secure Sockets Layer.* Previous name for Transport Layer Security (TLS).

TDES
: *Triple DES.* See 3DES.

TLS
: *Transport Layer Security.* The current name and standard definition that provides confidential and authenticated transmission of data over insecure channels.

Chapter 13

Indexes

13.1 Function index

13.2 Subject index